Shanky Technologies

PPL
POKER PROGRAMMING LANGUAGE

User Guide

Edition: November 8, 2023

bonus bots .com

Table of Contents

1. First Things First

Let’s get one thing straight: You don’t need to customize our poker bots! We are not selling a blank platform that requires programming. Our Holdem Bot is a fully functional software program that plays good poker right out of the box, and has many ready-made playing profiles targeting different game types already available.

Many users have reported high final table finishes in large 300+ field MTT’s completely unassisted, including plenty of first place wins, using the basic profiles that are already available. In addition, we have received multiple reports from users who have gotten a piece of large bad-beat jackpots using our bot. Many of our forum members have reported cashing a $500+ signup bonus while they slept. There have been graphs posted in our forum showing nice up-trends over many thousands of hands in cash games. All without any custom profile coding.

But there are always those who want to be able to tweak the bot to play the way they want it to, and that is why we developed PPL (Poker Programming Language). With it you can fully customize the play of our poker bots, whether making a few adjustments here and there or creating an entire set of playing instructions from scratch (or anything in between). If you are a researcher at a university it should be a simple matter to create a parser so you can use PPL in your project.

You **don’t need any computer programming training** to become proficient in PPL. It’s very easy to pick up if you are good with a PC and understand common poker terminology. Reading this guide – plus a little practice – should have you coding away to your heart’s content in no time.

2. Introduction

The Shanky Technologies poker bots are the most sophisticated software programs of their kind ever to become available to the public. In spite of their sophistication, they have always been extremely user-friendly.

Originally our bot was only customizable with the simple **option menu** on the bot menu itself, which contains many settings the user can adjust to have the software play in a specific way for different game environments. Many large tournaments have been won completely unassisted this way, many deposit bonuses have been cleared, and much rakeback earned.

As our products developed and continually improved, we eventually reached a plateau where we really couldn’t improve upon their play by adding more new features and user-options. The only logical path forward was to move in a direction of making the bot totally user-configurable, without removing the already proven successful programming and user-option list. This was not an easy task, but that has never stopped us before. There are literally thousands of possible poker situations, and hundreds of variable elements which create those situations.

What we came up with as a solution is a simple programming language that anyone can use, which includes the variables that our bot uses to categorize the information it gathers from the poker table. The code is easy and the list of variable elements and action commands is short. The purpose of this guide is to show you how to use it. Thus we believe we have kept the product user-friendly while greatly expanding upon its possible playing styles.

This also opens up a whole new community of users who enjoy creating, trading, and sharing their own custom profiles (even selling them if they want). If this sounds like fun, be sure to join our forum so you can get in on all the action.

Profile Format

Custom profiles are created and saved as simple text files. Please make sure you have downloaded the Holdem Bot and have read **sections 1 through 3 of the Holdem Bot User Manual** before proceeding any further in this guide. You should also have at least played around with the 200 free hands a little, and have practiced loading alternative profiles. Section 2 of the Holdem Bot User Manual explains the use of profiles. If you need a profile to practice loading, try the MTT version of the Doodle profile, which you can get here:

www.bonusbots.com/mtt.txt

You will need to copy all of the above code, paste it into an empty text profile, and save it in accordance with the instructions in section 2 of the Holdem Bot User Manual in order to load it into the bot.

You can use Notepad, the text editor that comes with Windows, to open, tweak, create, and paste in, and save profiles. If you download an existing text profile from our support forum and then double-click to open it, it is likely Notepad will automatically open it on your PC. If not, you should manually open Notepad and then use that to open any saved profile. Notepad is probably located in the *Accessories* folder on your Windows start menu. (Please do not use Word or Wordpad for accessing or editing profiles!)

We would, however, highly recommend that you download an enhanced text editor such as **Notepad 2** (which is free, just Google it) and use that instead. Regular Notepad that comes in Windows can mis-reference codelines, plus enhanced text editors such as Notepad 2 have the codeline numbers conveniently placed on the left of the profile.

2.1 Beep Feature = Fold or Beep Bot

Before we get into the PPL instructions in section 3, I want to mention a feature that our bots now have which appeal to an entirely new market segment. One of the actions now included in PPL is Beep. Beep means just what it sounds like: instead of calling, raising, checking, or folding, your computer will beep at you to get your attention so you can play the hand out yourself.

What this means is that instead of playing poker for you, you can have the software folding all the hands you don’t want to play on multiple tables and then beep at you (and bring the active table to the front) whenever a playable hand shows up so you can play it yourself. This makes the tedious task of multi-tabling much less taxing on your mental faculties.

To create this configuration you just need to follow the directions below for customizing the bot and create a simple piece of code that lists the hands you want to play and has the action set as beep. This is by far the easiest custom programming job. I will get you started with a template for our Holdem Bot below:

```
custom
preflop
```

```
When hand = AA beep force
When hand = KK beep force
When hand = QQ beep force
When hand = JJ beep force
When hand = TT beep force
When hand = 99 beep force
When hand = 88 beep force
When hand = 77 beep force
When hand = 66 beep force
When hand = 55 beep force
When hand = 44 beep force
When hand = 33 beep force
When hand = 22 beep force
When hand = A suited beep force
When hand = AK beep force
When hand = AQ beep force
When hand = 67 suited beep force
When hand = xx beep force


When others
 When others fold force


flop
When others
 When others beep force

turn
When others
 When others beep force

river
When others
 When others beep force
```

Simply open a new, empty text file and copy and paste all of the above code in it. You will need to complete it by copying the third to last preflop line (shown in red), and pasting it underneath that line over and over again then replacing the xx with all the other hands you want to play. You will need to *read in* the custom profile you have created every time you start the bot by choosing the *Holdem: Read Profile* menu item and specifying the name of the file which contains the above text.

See how easy that is? You now have a piece of poker software that folds all the unplayable hands for you automatically, and alerts you when you have a playable hand. Of course you can also program it to raise for you whenever you have AA or KK as well, call whenever you have AK and there has been exactly one raise less than 10 big blinds in size, and anything else you want automatically done. So you can configure a bot that plays most hands for you and only beeps for the more difficult discretionary type decisions. It’s up to you – the possibilities are endless!

2.2 Simple Fold or Push Bot

Here’s a profile for our Holdem bot that will shove or fold pre-flop. The hands for which the bot shoves can be modified easily. Some people might want to use a profile similar to the one below for playing certain Sit and Go tournaments, such as “super-turbos.’ To create this configuration simply create a text file which contains the code below and load it into the bot using the *Holdem* > *Read Profile* menu item.

```
custom
preflop

When hand = AA RaiseMax force
When hand = KK RaiseMax force
When hand = QQ RaiseMax force
When hand = JJ RaiseMax force
When hand = TT RaiseMax force
When hand = 99 RaiseMax force
When hand = AK RaiseMax force
When hand = AQ suited RaiseMax force

When others fold force
```

You can complete this by adding additional hands in like manner, underneath the AQ suited line (but above the “when others fold force” line). See how easy that is? You now have created a piece of poker software that goes all in with the hands you want to play and folds all the rest.

Please note the above profile does not close out post-flop situations, so the default profile will kick in and play those for you as a backup, those times you get a free look at the flop in the big blind. If you really want to use a “push or fold” profile you should probably create post-flop instructions for those situations as well, which you will learn how to do in the following sections.

Now let’s teach you how to create custom codes from scratch that will do anything you desire.

3. PPL for the Shanky Technologies Poker Bots

By following these instructions you can override all of our bot’s default programming and even your own settings from the option menu to have it play as you instruct it for any situation. As you have seen by the above example, PPL is very intuitive. We developed this simple, logical language consisting of only a few parts and any experienced online poker player should take to it like a fish to water.

Most of the rest of this manual will teach you how to use PPL for our Texas Holdem bot, which is our most popular product by far. A few examples here and there are thrown in that pertain specifically for our Omaha bots, and it should be obvious which variables in the code are meant for use with that product. It is assumed that if you want to learn to code one of our Omaha bots, then you also understand how to play Holdem as well. Therefore this guide should enable you to custom-code any of our poker bots.

3.1 What about a Custom Programming Interface?

Nice idea. Right now, our bots do not have a graphical interface for using PPL. The code only needs to be added to a text file (otherwise known as a notepad file). You then read in the code into the bot using the *Holdem>Read Profile* menu item. The bot will parse the code while reading it and print any errors in the log window and in the *Holdem.log* file. If there are any errors the entire code is rejected.

You can have all your options settings in the same file as your PPL code. In order to do that, first save a profile. Go through the option menu and select the options you want for a certain playing profile. Then save the profile by going to *Holdem>Write Profile* on the bot menu. Change the name to something shorter and easy to type, and click *save*. As long as you didn't specify a different path it will be saved to the same folder you have all the bot download files in. This is explained in detail in our *Holdem Bot User Manual*. After you have saved a profile you can open it. Scroll to the bottom. Your customized PPL code simply needs to be pasted underneath what you see saved on that text file. In fact you can even just type your code there to begin with.

3.2 PPL Structure and Elements

We will first discuss the structure and elements of PPL. In later sections we will give various examples that will make the usage clearer. Don't run away after reading this section. The examples will show you how simple this really is.

PPL is structured as follows. The code consists of a series of *Statements*. Each new statement starts with the word *When* and is of the form:

```
When <Condition> <Action>
```

The <Condition> specifies a situation in which to take the specified <Action>. For example:

```
When Hand = AA Raise
```

The above tells the bot to raise when it is dealt pocket aces. In the above "Hand = AA" is the <Condition> and "Raise" is the <Action>. **See how simple it is.**

There is another *Statement* of the form:

```
When <Condition>
```

This is referred to as a 'When Condition without an action'. Such statements and their use are discussed in section 3.3.

3.2.1 Conditions:

Conditions are used to specify a particular situation including your hand, your position, the number of raises made, the bet or raise size etc. The complete list of variables that can be used inside conditions and their meanings is given in Appendix 1. This section describes the kinds of Conditions that can be specified.

Currently there are six kinds of <Conditions>. More types may be added later. The 6 types of <Conditions> are:

1. **Hand Specification Condition**: This kind of condition specifies the cards you have. For the Holdem bot this is of the form:

   Hand = <Card Specification>

   Or

   Hand = <Card Specification> <Card Specification>

   Where <Card Specification> specifies a single card. For Holdem up to 2 cards can be specified as a hand can have only 2 cards. For the Omaha bot up to 4 cards can be specified. Each <Card Specification> needs to specify the rank (2, 3, 4, 5, 6, 7, 8, 9, T, J, Q, K or A) and optionally the suite (C, D, S or H) and optionally whether or not the card is suited by using the keyword "suited". *Note: Please note that 10 is not valid, you must use T for the rank 10*

   Examples

   ```
   Hand = A
   ```

   Means a hand that contains an Ace (whether suited or not). Since we have not specified any other cards, the above <Condition> will be true whenever we have a hand that contains an Ace irrespective of the 2nd card.

   ```
   Hand = A suited
   ```

   Means a hand that contains an Ace any other card of the same suit.

   ```
   Hand = A K
   ```

   Means a hand that contains an A and a king (whether suited or not).

   ```
   Hand = A K suited
   ```

   Means a hand that contains an A and a king and the king is suited to some other card in the hand. For Holdem with only two cards in the hand, if the king is suited then both cards are the same suit. Therefore the above is the same as saying:

   ```
   Hand = A suited K
   ```

   Which means a hand that contains an A and a King and the Ace is suited. It makes no difference for Holdem but for Omaha with 4 cards in the hand, it is important to specify the 'Suited' keyword with the right card.

   ```
   Hand = Ac Kc
   ```

   Means a hand that contains an Ace of Clubs and a King of Clubs.

   ```
   Hand = A34
   ```

Means a hand that contains an Ace and a 3 and a 4. This specification will cause the Holdem bot to print an error message since you can only specify up to 2 cards for Holdem. However the above is a valid <Card Specification> for the Omaha bot.

Please note that you can also specify when your hand has any pocket pair or any two pair (for Omaha) or trips (for Omaha) by using the Boolean variables:

```
PairInHand
TwoPairInHand
TripsInHand
```

2. **Board Specification Condition**: This is similar to the Hand Specification Condition above and is used to specify the cards on the board. This only makes sense for post-flop rules. This is of the form:

    Board = <Card Specification> <Card Specification> …

    Where <Card Specification> specifies a single card. Up to 5 cards can be specified. The card specifications are done exactly the same way as in the Hand Specification above.

    Example:

    ```
    Board = A K Q
    ```

    Means a hand that contains an Ace, a King and a Queen (whether suited or not).

3. **Variable Comparison Condition**: This kind of condition is used to compare a *variable* to a *value* (numeric valued variables only). It is of the form:

    <Variable> <Comparator> <Value>

    For example:

    ```
    Raises < 2
    ```

    In the above "Raises" is the <Variable>, the "<" symbol is the <Comparator> and "2" is the "Value". This condition is true if the number of raises is less than 2.

    <**Variable>**: The full list of numeric valued variables which can be used in a comparison conditions are specified in Appendix 1. Some examples are:

    Folds – *number of folds on the current betting round*

    Checks – *number of checks by opponents in this betting round. The bots' own checks are not counted. This variable is reset to 0 when flop, turn or river cards are dealt so that it represents the number of checks for the particular betting round only.*

Calls – *number of calls by opponents in this betting round. The bots' own calls are not counted. This variable is reset to 0 when flop, turn or river cards are dealt so that it represents the number of calls for the particular betting round only.*

Bets – *number of bets by opponents in this betting round. The bots own bets are not counted. This variable is reset to 0 when flop, turn or river cards are dealt so that it represents the number of bets for the particular betting round only. The value can be either 0 or 1 since not more than 1 bet can be made in a particular betting round.*

Raises – *number of raises by opponents in this betting round. The bots own raises are not counted. This variable is reset to 0 when flop, turn or river cards are dealt so that it represents the number of raises for the particular betting round only.*

<Comparator>: A comparator can be one of the following with the usual meaning

=
<
>
<= (*no spaces and =< is not valid)*
>= (*no spaces and => is not valid)*

<Value>: A value is just any number such as *0, 4, 10, 73, 150, 1004, etc.*

4. **Relative Potsize/Stacksize Comparison Condition**: This is a special condition designed to allow you to compare the percentage of certain numeric valued variables to either the *potsize* variable or the *stacksize* variable (also both numeric valued variables) . This is of the form:

BetSize, TotalInvested, or AmountToCall <Comparator> <Value>% PotSize
Or
BetSize, TotalInvested, AmountToCall, or StartingStackSize <Comparator> <Value>% StackSize
Or
MaxCurrentOpponentStacksize <Comparator> <Value>% Potsize or StackSize
Or
Potsize <Comparator> <Value>% StackSize

…plus a few others (**see section 3.5.8**)

For example:

```
BetSize < 30 % PotSize
```

This condition would be true when the last Bet or Raise Size is less than 30% of the current Pot Size. Please note that the opponent's last bet is included in the pot size calculation with our poker bots, which makes these bet size vs. pot size calculations a bit tricky (more about this in the post-flop coding section).

Similarly this can also be used to compare the Total money the bot has invested into the pot for this hand as a percentage of the PotSize or StackSize.

For example:

TotalInvested > 100 % StackSize

The above condition would be true when the total money invested into the pot in this hand is more than 100% of the bot's remaining stack. That is, the remaining stack size is less than the amount already invested in this pot. You may want to program the bot to call all further bets in this situation.

Another popular use is to identify the depth of a live opponent's stack size using MaxOpponentCurrentStacksize.

For example:

MaxOpponentCurrentStacksize > 100 % StackSize

The above condition would be true when every opponent still in the hand currently has at least as many chips as you do. Similarly:

MaxOpponentCurrentStacksize < 20% StackSize

…means every opponent still in the hand has a short stack compared to yours, less than 20%.

5. **Position Conditions**: This is a special set of conditions designed to specify the bot's position. This can be of the form:

   In <Position>

   Where <Position> can be BigBlind, SmallBlind or Button.

   For example:

   In BigBlind

   This condition would be true when the bot is in BigBlind. Note that the above conditions can be used post-flop also. When used post-flop they refer to the position of the bot before the flop. Note that the *StillToAct* Variable can also be used to specify position. Preflop, the following Condition is the same as *In Button.*

   StillToAct = 2

   However after the flop or after the betting has come back to the bot after a raise *in Button* will still be true if the bot is on the button but *StillToAct = 2* will be true only if there are 2 players left to act after the bot.

   Another way to specify position is to use conditions of the form:

   Position = <Relative Position>

Where <Relative Position> can be First, Middle or Last.

For example:

```
Position = First
```

This condition would be true when the bot is first to act. Note that *position = last* is the same as *StillToAct = 0* if this is the first time the bot is betting in a particular round. However if betting comes back to the bot after a raise then *StillToAct* will always be 0 since everybody in the game has acted at least once but *position = last* will be true only if the bot was the last to act in the initial round.

6. **Boolean Variables**: This condition is specified simply by giving the name of the variable and is of the form

```
<Boolean Variable>
```

It is a simple true or false variable. The full list of Boolean Variables is specified in Appendix 1. A few examples are given below.

Others is a special catch-all Boolean variable which is always *true* irrespective of the state of the game.

For example the statement:

```
When Others fold
```

…will cause the bot to always fold.

Other examples of Boolean Variables include *FlushPossible* (which is true when there are at least 3 cards of the same suit on board so that a Flush is Possible). Similarly *HaveFlush* is a Boolean Variable which is *true* if we have a made flush. User defined variables are Boolean variables that start with the letters 'user'. These are explained in section 3.2.5.

3.2.2 Operators:

A condition can be composed of other Conditions. Conditions can be combined using the boolean *operators "And", "Or"* and *"Not".* They can also be grouped together using parenthesis.

For example:

```
When hand = QQ and raises = 0 RaisePot

When (hand = AA or hand = KK) and raises >= 1 RaiseMax
```

'*Not*' and '*And*' and '*Or*' have their standard meanings. '*Not*' has a higher precedence than '*And*' which has a higher precedence than '*Or*'.

Not <Condition>

Is true only when <Condition> is false and vice versa.

<Condition> And <Condition>
Is true only when **both** conditions are true.

<Condition> Or <Condition>
Is true when **either** condition is true.

3.2.3 Actions:

An action tells the bot what to do when a certain condition is satisfied. The bot currently supports the following actions:

Beep - *computer will beep at you but take no action on the hand – if you do not respond you will time out*

Call – *check if possible else call*

Play - *same exact action as 'call' above, use either*

Raise - *generic Raise action. The size of the raise depends on available buttons for the particular poker room, the user-defined custom raise size, if any (see Custom Bet Sizing below), and certain option settings. Will raise minimum preflop unless the "Make Pot Sized Raises for" option is set or a custom bet size is specified.*

RaiseMin - *will raise minimum*

RaiseHalfPot - *will raise pot or max if half pot button not available and raise min if no other button available*

RaisePot - *will raise max if pot button not available and raise min if no other button available*

RaiseMax - *will raise pot if max button not available and raise min if no other button available*

Fold - *will check if available else fold*

Bet - *will bet if available otherwise will Call. The size of the bet depends on available buttons for the particular poker room, the user-defined custom bet size, if any (see Custom Bet Sizing below), and certain option settings. Note that Bet is never available pre-flop.*

BetMin - *will bet minimum if bet option is available otherwise will call. Note that Bet is never available pre-flop.*

BetHalfPot - *will bet pot if half-pot button is not available otherwise will bet max if available. Will bet min if no other bet option is available. Will call if bet option is not available. Note that Bet is never available pre-flop.*

BetPot - *will bet pot if button is available otherwise will bet max if available. Will bet min if no other bet option is available. Will call if bet option is not available. Note that Bet is never available pre-flop.*

BetMax - *will bet max if button is available otherwise will bet pot if available. Will bet min if no other bet option is available. Will call if bet option is not available. Note that Bet is never available pre-flop.*

SitOut - *will sit out of the game and discontinue playing. Once invoked there is no way to sit back in.*

User Defined Variable – *A user defined variable (explained in section 3.2.5 below) is a special action that can appear wherever an action can appear. If the action for a matching when condition is a user defined variable, the result is simply to set the value of the user defined variable to true.* ***Unlike other actions, execution continues to the next PPL statement after setting the variable to true****. If any other action is executed no more PPL statements are looked at.*

The "***force***" keyword can be used after an action. Using *force* causes the bot to ignore all user settable option settings that may otherwise cause it to behave differently. For example:

```
When hand = AQ Call force
```

Will cause the bot to call when holding AQ irrespective of the number of raises even if the option to *Fold for Pre-Flop Raises for AQ, AJs, & KQs* is set. If the *force* keyword is omitted then the actual action will depend on the option settings. Therefore it is recommended to always use the keyword *force* when specifying actions.

3.2.4 Custom Bet Sizing:

The *Bet* and *Raise* actions can be combined with a number or a % figure to specify a custom bet size.

Bet/Raise by a Number – if you specify a number after the action *Bet* or *Raise* the bot will bet or raise by the number specified. Please note that the number is number of big blinds, not a dollar amount.

Examples

Bet 5 force (bets 5 big blinds, note that *Bet* is never available preflop).

Raise 5 force (raises by 5 big blinds, note that *Raise* is always raise by, not raise to).

Bet/Raise by a % – if you specify a number with a % sign after the action *Bet* or *Raise* the bot will bet or raise by the **percentage of existing pot size** specified.

Examples

Bet 70% force (bets 70% of the pot, note that *Bet* is never available preflop).

Raise 150% force (raises by 150% of the entire existing pot size, note that *Raise* is always raise by, not raise to).

If a bet or raise size is specified that is more than the current stack size, then the amount is reduced to the current stack size. If a bet/raise size is specified that is more than the pot size in PL games, then the amount to be bet is reduced to the pot size.

Our bots execute the custom bet sizing commands by typing the bet size specified into the bet amount window at the poker table and then clicking on the Bet button, just as you would. Note that this sometimes results in odd-looking bet amounts and cannot be helped. If the bet size is over $10 it is rounded to the closest whole dollar amount in an effort to appear as a more natural betting amount.

3.2.5 User Defined Variables:

This is a Boolean-style variable that the user defines. Any normal codeline that defines a situation using any of the existing PPL elements can set a user defined variable as the action, instead of taking a normal action. It can even include previously set user-variables. Simply state a user-variable instead of an action at the end of the codeline. Once set, a user-variable is used as a Boolean variable, being read as either true or false in the codeline where it appears. Setting a user-variable is not like a normal action (raise, call, check, bet, fold, beep, etc.) where the bot stops reading code and executes the action. When the action is setting a user-defined variable the bot keeps reading code until a matching condition with a normal action is found.

User defined variables start with the word 'user' followed by one or more letters, numbers, or the underscore character. "Usergroup1," "user_a,", and "UserMinRaise3" are examples of user defined variables. All variables are case insensitive as usual. So "userabc", "USERABC" and "UserAbc" are the same variable and thus are interchangeable.

All user defined variables are set to false when cards are dealt for a new hand. A user defined variable is "turned on" (or "set") when it appears as the action clause of a matching when condition. It can then be used just like any other Boolean-valued variable as part of a condition. Once set, the variable retains its 'true' value until the cards are dealt for the next hand. Therefore user-variables that are set during an early betting round, such as pre-flop or the flop, can be used on later streets such as the flop, turn or river.

The user defined variable is useful for defining specific situations on an earlier betting round that are difficult to define later, such as your exact pre-flop table position or that of an opponent. They are also handy for abbreviating longer codelines that you want to use later in the code, such as a group of starting hands. Careful attention should be given to their strategic placement in the code structure, understanding that the line which sets the user-variable must be read first in order to turn it on, and knowing that when a normal action is executed no further code is read.

Many creative uses for the user-variable are possible, limited only by your own imagination. Here are some examples to get your wheels turning, so you can understand the possibilities better. (Keep in mind some of these examples may only become clear to you after you digest the rest of this manual.)

Examples

```
When  raises  > 2 and not (hand = AA or hand = KK or hand = QQ or hand = AK) User3Raises
When User3Raises fold force
```

In the above example the user defined variable called 'User3Raises' is set whenever the opponent raise count is more than 2 and we don't hold a premium starting hand. The next statement then causes a fold to happen when that condition is found true. This is a simple example, just meant to explain the abbreviation nature of the user-variable.

Here is an example that shows the importance of logical placement of the user-defined variable in your code:

```
preflop
When UserKK raisemax force
When hand = KK UserKK
When UserKK raisepot force
```

The above pre-flop code will cause the bot to RaisePot the first time the bot acts when holding KK; however if someone raises afterward and the action comes back, it will then go all-in by executing the RaiseMax action. It is important to understand why the RaiseMax action above can only happen if there is a raise behind the bot after it acts the first time. The 'UserKK' variable is not set yet on the RaiseMax action line the first time around, so it is not found as true. The next line actually sets the variable. If somebody reraises after we act, the variable will be then have been set and will be found true for the RaiseMax codeline, which is read first.

In this next example we will define absolute pre-flop table position that we can use later in the hand:

```
preflop
when stilltoact = 8 UserUTG
when stilltoact = 7 UserUTG1
when stilltoact = 6 UserMid1
when stilltoact = 5 UserMid2
when stilltoact = 4 UserMid3
when stilltoact = 3 UserCutoff
```

…now you can specify your exact starting table position anywhere in the code, on any betting round. For example later on the preflop round if you use this as part of a statement:

```
when raises = 1 and botslastaction = raise and  (in button or UserMid3 or UserCutoff)….
```

...you know that you got reraised after raising from a late position, so it is more likely that the opponent is reraisng weak than if you raised from an earlier position.

Here is an example that helps you identify the absolute table position of your opponent:

```
preflop
when raises = 1 and lastraiserposition >= 6 UserEarlyRaiser
```

…now you know an opponent opened strong from early position, so you can use this to refine plays later in the hand, for example:

```
flop
when havetoppair and havebestkicker and not (paironboard or straightpossible or havetwopair or haveflushdraw) and position = last
and bets = 1 and raises = 1 and botsactionsonthisround = 1 andbotslastaction = raise and UserEarlyRaiser fold force
```

…this flop codeline describes a situation where the opponent who raised in early position preflop then bet the flop, and you raised holding TPTK, and then they reraised. It is very likely they have a big overpair like AA or KK here, so you can feel better about folding. Without the user-defined variable we couldn't know with certainty on the flop that the opponent was an early position pre-flop raiser.

This next example shows an easy way to create pre-flop hand groups that may allow you to reduce the amount of code used later when repeating these hand groups:

```
preflop
when (hand = AA or hand = KK or hand = QQ or hand = AK) UserG1
when (hand = 22 or hand = 33 or hand = 44 or hand = 55) UserBabies
when (hand = 5d6d or hand = 67 suited or hand = 78 suited or hand = 89 suited) UserSCons
```

…which could then be used in the following manner:

```
when stacksize < 15 and UserG1 raisemax force
when (UserBabies or UserSCons) and amountocall < 23% potsize call force
```

Creating preflop hand groups incorporating already-set user-variables can be done like this:

```
when (hand = AA or hand = KK or hand = QQ or hand = JJ or hand = TT) userlist1
when userlist1 or (hand = 99 or hand = 88) userlist2
when userlist2 or (hand = 77 or hand = AK) userlist3
```

Here is an example of identifying a limp-raiser from UTG or UTG+1 and folding for fear of AA or KK:

```
preflop
when raises = 0 and calls >= 1 and firstcallerposition >= 7 UserTrap1

when position = last and botslastaction = raise and UserTrap1 and not (hand = AA or hand = KK or stacksize < 12 or totalinvested  > 80% stacksize) fold force
```

Finally, let's look at an example of setting the user-variable on a later betting round. The code below will identify a situation where on the Turn the board is all low cards, and then a face-card or ace lands on the River and the opponent bets:

```
turn
when not (board = A or board = K or board = Q or board = J or flushpossible or straightpossible) UserLowTurn

river
when bets = 1 and raises = 0 and (board = A or board = K or board = Q or board = J) and UserLowTurn and not (amounttocall  < 30% potsize or amounttocall  <= 2 or havetoppair or haveoverpair or havetwopair or havetrips or haveset or havefullhouse or havequads or haveflush or havestraight or nobettingonturn) fold force
```

…in this case we decided to fold hands below top pair in value if the bet size was not small.

While the user defined variable can really come in handy, keep in mind that it is a bit of an advanced feature and using it is not necessary for creating good profiles. Plenty of winning profiles were created before we developed it. We suggest you get familiar working with the standard variables first, before dabbling in user-variables, and incorporating them only as you see fit. Most of the better profiles only use a few of these, if any.

3.2.6 Opponent Name Variable

This is a numeric-valued variable with user-created custom values that can be used to identify the screen name of individual opponents you are playing against. See the appendix for the current list of poker rooms this feature is restricted to. (Note: as of December 2016 this only works at the WPN sites, America's Cardroom, etc.) The variable is stated as:

Opponent =

It will accept values consisting of any combination of these characters:

Upper case letters
Lower case letters
Numbers
Dashes –
Underscores _

The variable is true if there is a match for your value with any opponent screen name in the current hand who still has cards in front of them. It can be used on any betting round, and user-defined variables can be set from long codelines identifying multiple screen names. However, the best use is probably on post-flop betting rounds, or second-orbit situations preflop, when you can couple it with opponents = 1. Notice how the opponent**s** = variable is only one digit different, so be sure not to confuse it with the opponent = variable.

Using the opponent name variable provides a way for you to utilize stats gathered on opponents from tracking software such as Poker Tracker, Holdem Manager, etc. You can also simply use screen names from opponents you have personally observed and want to play a certain strategy against. By being able to use their screen name as a variable, it doesn't matter which stat software you use, or even if you acquired a list from somebody else who gathered the stats. You also can run the bot without the stat-gathering software running in the background, which would be quite taxing on PC resources.

Examples

```
when (opponent = egor or opponent = Mad_Scientist) fold force

when opponents = 1 and raises = 1 and (opponent = philivey or opponent = MikeSexton or opponent = uni-bomber3 or opponent =
1982_ozzy or opponent = durrrrr or opponent = mad-as-hell) raisepot force

when opponents = 1 and (opponent = pilotbatch or opponent = Dman or opponent = Chevy25 or opponent = OurManinLondon or
opponent = pingu or opponent = ShankyBot_26 or opponent = Bonus-Bots-rock) Usershark

turn
when bets = 0 and raises = 0 and Usershark bet 75% force
```

3.3 Writing a Custom Profile

This section takes you through the process of writing a custom profiles using the elements of the PPL language discussed in the previous section. The explanations below are supported by examples.

3.3.1 Start with the Word *custom* on One Line

The custom instructions must start with the single word custom on one line by itself, all in lower case letters, underneath where all your chosen option settings display on your profile.

After that **case is not important** and spaces and line breaks are not important either, with the notable exception of comments (see below) where they are ended by a line break.

So the statement:

```
When hand=AK Suited Raise force
```

is the same as:

```
WHEN HAND = a k suited
Raise force
```

Consequently there are different ways to code the same desired actions, and different ways of using the language elements to create valid code which says the same thing. The important thing is to understand how to piece the elements together to make valid code. If you are trying to code more complicated scenarios involving the opponent count, position, last bet size, and such, you are going to want to create the desired actions with the least amount of code.

3.3.2 Use *preflop*, *flop*, *turn* or *river* on One Line to Start for Each Betting Round

The betting round must be identified with one of these terms to start the coding for each betting round.

```
preflop
flop
turn
river
```

Therefore if you are including preflop custom instructions your code would start with:

```
custom
preflop
```

…and when you are done with the preflop section you would enter the line

```
flop
```

…to start coding the flop situations.

3.3.3 End your code with *When others* + *when others fold* if you Want to Kill All Default Programming

After you have completed your code for all the conditions that you want customized for a betting round, you can tell the bot to only play the situations you have customized for and ignore all the option settings and default programming (if you want). This is done by adding the following two lines at the end:

```
When others
When others fold force
```

You would then state the one line word to signify the start of the next betting round code, if applicable.

However in most cases you probably don't want to use that with the Shanky Technologies poker bots, as these programs play a good poker game by default. That is, you probably only want to customize a few situations and have the bot play by your option settings for everything else. But if you want to write an entire set of instructions for a particular betting round you would use that at the end. An example might be if you are creating a *Fold or Go All In* type profile for playing Turbo SNG's.

3.3.4 Use the Word *force* after your Action

User modifiable option settings can override the action specified. If a conflict is found, it will choose the action indicated by the option setting unless your custom code has the word *force* after the action. Since there is usually not much point in custom-coding something that will be overridden by your option settings, all the code examples in this manual include the word *force* after each action command.

3.3.5 Bet Size Value

The bet size is always stated in terms of **number of big blinds**, regardless of which betting round it is. So if you want to fold pre-flop for raise sizes above 4 big blinds your code snippet might look like this:

```
When (hand = JT suited or hand = QJ suited)
   When raises = 1 and BetSize <= 4 call force
   When raises = 1 and BetSize > 4 fold force
   When raises > 1 fold force
```

This gets a little more complicated on later betting rounds. You have to decide how big of a bet to call based on the original big blind size. Fortunately we provide a *PotSize* variable to help make that calculation for you, along with a % comparator which can be applied to the pot size value post-flop. (Note that the above piece of code introduces multiple *When Condition* statements.)

3.3.6 Statements

I am using the term *statement* a lot in this guide. The PPL code that you will create consists of a series of statements. What I mean by a statement is a defined action or series of actions which achieves a desired result for a specific condition.

Statements can be long or short. **Statements always start** with the word When. Stating the action is always done with one word (RaisePot, fold, etc). However conditions usually require multiple words including comparator symbols and values.

The condition that you probably want to start with is a **hand or a group of hands** that you want to set actions for. It is also possible to start with one of the three position conditions included in the language, or you could start by using a set of variables to define your own custom condition. But let's face it, there is no position (or other set of variables outside your hand) that you are going to want to play the same way all the time irrespective of what cards you are holding. Also, coding by position is much more cumbersome than coding by hands. Let's look at a couple of examples to see what I am talking about.

```
When In Button and raises = 0 and calls = 0 raise force
```

If you are a borderline maniac who always wants to raise on the button when folded to no matter what cards you have, you might want to start your code with that statement. Now let's look at an example of a statement for coding by custom-defined position, in this case under the gun:

```
When raises = 0 and calls = 0 and folds = 0 and StillToAct >=7
   When (hand = AA or hand = KK or hand = QQ or hand = AK) RaisePot force
   When (hand = AQ suited or hand = AJ suited or hand = JJ or hand = KQ suited) call force
   When others fold force
```

The above statement will tell the bot to only play the specific listed hands when under the gun in full ring games with 7+ live opponents behind you. Coding for position like this is something you might want to do if you have some regular settings on our bot that you like, except you just want to change how it plays in certain positions. For example if you just want it to tighten up when in the first two positions.

Coding for second position (UTG+1) is slightly more complicated though, because you now have to account for different possible actions by the UTG player.

```
When raises = 1 and calls = 0 and folds = 0 and StillToAct >=6
   When (hand = AA or hand = KK) RaisePot force

When raises = 1 and BetSize < 8 and calls = 0 and folds = 0 and StillToAct >=6
   When (hand = QQ or hand = or hand = AKs) RaisePot force
   When (hand = AQ suited or hand = JJ) call force
   When others fold force

When raises = 1 and BetSize >= 8 and calls = 0 and folds = 0 and StillToAct >=6
   When (hand = QQ or hand = or hand = AKs) call force
   When others fold force
When (raises = 0 and calls = 1 and folds = 0) or (raises = 0 and calls = 0 and folds = 1)
    When (hand = AA or hand = KK or hand = QQ or hand = AK) RaisePot force
    When (hand = AQ suited or hand = AJ suited or hand = JJ or hand = KQ suited) call force
    When others fold force
```

As you can see it now takes four statements to code for the exact same eight hands in the UTG+1 position where it only took one statement to code them in the UTG position. As you move down in position it becomes exponentially more complicated, especially since you will be adding more hands as you go.

For that reason we would highly recommend coding by hands instead of by position, unless you just want to do something for one or two positions and play using the bot's option settings the rest of the time. It's much easier to state a group of hands and then cover all possible positional situations below them. Here is an example:

```
When (hand = 77 or hand = 88)
    When raises > 1 fold force
    When raises = 1 and BetSize > 5 fold force
    When raises = 1 and BetSize < 5 call force
    When opponents <=2 and calls <= 1 RaisePot force
    When opponents <=2 and calls > 1 call force
    When opponents > 2 call force
```

There is a perfectly good piece of code for handling 88 and 77 in all positions and situations in a No Limit game. See how clean it is, and how little time it took to write? For the rest of this user guide we will be assuming that this is the way you are coding, that is defining specific blocks of hands and then specifying the actions for all the different conditions you want to define for those hand blocks.

3.3.7 Start Multiple Lines in Statements with *or* or *and*

As you already know a statement can be just one line. However sometimes you may want to code multi-line statements that use the word *and* or *or* to further define the condition on subsequent lines before stating the action. Here is an example:

```
When hand = 8c 7c and In BigBlind
   and raises = 1 and calls = 0
   and BetSize < 7
   RaisePot force
```

What you have there is a statement that must include all three lines for the condition to be defined. You only want to re-raise a lone opponent who raised when you are in the big blind with exactly 87 of clubs and his raise was under 7 big blinds – you want no other players in the pot. This type of situation is when you use the word *and* to join conditions **all of which must be present** before taking the desired action.

Another example is:

```
When hand = AA
    Or hand = KK
    Or hand = QQ
    RaisePot force
```

Here the bot will *RaisePot* whenever it has either AA, KK **or** QQ. Either condition can be satisfied before taking the desired action.

Please note that these types of statements can just as easily be put all on one line, like this:

```
When hand = 8c 7c and In BigBlind and raises = 1 and calls = 0 and BetSize < 7 RaisePot force

When hand = AA or hand = KK or hand = QQ RaisePot force
```

3.3.8 When Condition Without an Action

We have already exhibited several examples of this type of statement in prior sections of the manual.

Pay close attention to the next paragraph so you learn how to use this kind of a statement correctly!

There are two different types of statements that start with *when*. One is a *when condition without an action*, and the other is a *when condition with an action*. Simple codes can consist entirely of *when conditions with an action,* such as the beep code template given earlier. A *when condition without an action*, however, will be connected to **each and every** when condition with an action below it. This is so important to understand that I am going to try and drill it through your head with a red-letter rule:

A *when condition without an action* will be separately ANDed to every *when condition with an action* below it UNTIL another *when condition without an action* is found - which starts the process over.

The previous code given for playing 77 and 88 is an example of that. Here is another one:

```
When hand = AT
   when raises = 0 and calls = 0 and opponents <= 5 raise force
   when raises = 0 and calls = 0 and opponents = 6 call force
   when raises = 0 and calls = 0 and opponents > 6 fold force
   when raises = 0 and calls = 1 and opponents >= 3 call force
   when raises = 0 and calls = 1 and opponents < 3 raise force
   when raises = 0 and calls >=2 call force
   when in BigBlind and raises = 1 call force
   when raises >= 1 fold
```

In the above code the first line (*When hand = AT*) is a *when condition without an action*. It is ANDed with all the subsequent statements starting with a when which do include an action. **Any one** of the conditions listed in the statements below the first (which defines the hand group) can be present and acted upon. This is the best way to code for the more common playable hands, and it's a good idea to cover all possible situations (as in the above example) to make sure they get played the way you want.

The "UNTIL another *when condition without an action* is found" part of the red letter rule above means that you **cannot have multiple when conditions without an action**.

Here is an example of an invalid pre-flop code.

```
When hand = TT
```

```
  When raises = 1
  When BetSize > 10
  Fold force
```

The above code is invalid and will be rejected by the bot with an error message.

To code this statement properly, do this:

```
When hand = TT and raises = 1
  When BetSize > 10 fold force
```

See the difference? The when condition without an action is followed by a when condition with an action. But you can have multiple *when conditions with an action* in succession as the *when condition without an action* is live until another *when condition without an action* is found. Here is a better example of a valid pre-flop code:

```
When hand = TT
  When raises = 1 and BetSize > 10 fold force
  When raises = 1 and BetSize < 10 call force
  When raises > 1 fold force
  When raises = 0 and calls <=2 RaisePot force
  When raises = 0 and calls >2 call force

When hand = 99
  When raises = 1 and BetSize > 7 fold force
  When raises = 1 and BetSize < 7 call force
  When raises > 1 fold force
  When raises = 0 and calls <=1 RaisePot force
  When raises = 0 and calls >1 call force
```

You see there how the statement starts with a *when condition without an action* (When hand = TT) followed by *when conditions with an action* which cover all possible scenarios for the hand. Then the next hand group starts with a *when condition without an action* (When hand = 99). That is the most efficient way to code by hand or hand group.

Be careful! Once you have started a *When Condition without an Action,* it remains active until a new one is given. This means that once you start coding this way, you need to keep coding this way for the remainder of the betting round. Most codes that we have developed are a mix of this type of code block along with some simple 1-line codes that are complete conditions with an action. Those complete simple 1-line codes need to be placed at the beginning of the betting round code, before you start using *When Conditions without an Action.*

[Note: If you just want to make sure that you go back to the situation where no *When Condition without an Action* is active you might want to insert the statement:

```
When others
```

…by itself. This causes the condition *“others”* to be active for all subsequent statements. This is the same as nothing being active since *“others”* is always true and ANDing with true does not do anything.]

Also, for nice clean readable code you should get in the habit of using line breaks and indentations like you see me doing. Even though it doesn't matter as far as the validity of the code goes, it makes it easier to read and therefore is **much** easier to go back through and edit.

3.3.9 If a Statement Contains Both *and's* and *or's*, Surround the *or's* in Parenthesis

Parenthesis are supported in the language and the above rule must be followed. This is critical. If you ignore this your code will not work as intended, because *and* has a higher priority than *or*. That means your code might get broken up the wrong way if you mix *and* and *or* in a statement. For example:

Invalid

```
When hand = KQ suited or hand = AJ suited and raises = 1 and BetSize < 10 call force
```

If what you wanted was for the bot to call with KQ suited or AJ suited only when the raise count is 1 and the raise was less than 10 big blinds in size, that is **not** what you will get with the above code. Most likely the bot will do what you want with AJs there, but when holding KQs it will call all raises regardless of how many there are or what size they are.

You need to be more precise in telling it what to do. By following the parenthesis rule you will stay out of trouble. However make sure you **surround the entire condition** with parenthesis, not just the values!

Here is an example of the wrong way to fix the above line of code using parenthesis:

Invalid

```
When hand = (KQ suited or hand = AJ suited) and raises = 1 and BetSize < 10 call force
```

That is invalid because the entire condition is not surrounded. Here is the correct way to fix it:

```
When (hand = KQ suited or hand = AJ suited) and raises = 1 and BetSize < 10 call force
```

Now the bot will do what you want. Study that last line of code for a bit so you get it right in your mind.

Remember to only include the *or's* in parenthesis – if you surround both *or's* and *and's* in them you will be right back where you started. For example:

Invalid

```
When (hand = KQ suited or hand = AJ suited and raises = 1) and BetSize < 10 call force
```

…is also incorrect. Now the bot will ignore the raise count for KQs again. Only surround the *or's* as in the previous correct example.

3.3.10 Code is Read in Sequential Order

This concept is also necessary to grasp in order to get the bot to take your desired actions for all your conditions. **The bot reads the code from top to bottom and executes the first matching command** for the situation. The reason this is important is that there are going to be some conflicting statements in the custom code sometimes, most notably suited hands verses unsuited hands and positional exceptions (especially for the blinds). This is because unsuited hands include suited hands but suited hands do not include unsuited hands, and there are hands that you might want to call a raise with only when you are in the blinds or on the button. To create valid code which does what you want, remember these three rules:

- Code Suited Hands First
- Code Positional Exceptions First
- Post-flop Code for Made Hands Before Draws

For example, if you tell the bot to fold QJ off-suit for a small raise but call a small raise with QJ suited, you need to put the line of code with QJ suited first. Otherwise the bot will find the line to fold for QJ first and fold your QJ suited hand because that is a QJ hand. If you put QJ suited first it solves the problem because after a match is found the bot quits searching the code.

Here is the wrong way to do it:

```
When hand = QJ and raises = 1 fold force
When hand = QJ suited and raises = 1 and BetSize < 7 call force
When hand = QJ suited and raises = 1 and BetSize  >=7 fold force
```

The bot will never find the QJ suited line in the above code and execute the action because it will find the QJ line first and fold. To make this code valid just reverse the order of the lines to:

```
When hand = QJ suited and raises = 1 and  BetSize < 7 call force
When hand = QJ suited and BetSize >=7 fold force
When hand = QJ and raises = 1 fold force
```

(Remember that line breaks and spaces don’t matter – I just insert them to make the code easier to read.) This same concept holds true for the blinds.

Here is the wrong way to code blind exceptions:

```
When hand = A9 and BetSize < 6
  When raises >= 1 fold force
  When raises = 0 and calls =>2 call force
  When raises = 0 and calls = 1 fold force
  When raises = 0 and calls = 0 and opponents >= 4 fold force
  When raises = 0 and calls = 0 and opponents < 4 raise force
  When In SmallBlind and calls >=1 call force
  When In BigBlind and raises = 1 call force
```

The bot will never make that big blind call and will not make the small blind call when calls = 1 because it won't get that far. The earlier *when conditions with an action* will tell the bot to fold first. To correct this code, move the lines around like this:

```
When hand = A9 and BetSize < 6
   When In SmallBlind and calls >=1 call force
   When In BigBlind and raises = 1 call force
   When raises >= 1 fold force
   When raises = 0 and calls =>2 call force
   When raises = 0 and calls = 1 fold force
   When raises = 0 and calls = 0 and opponents >= 4 fold force
   When raises = 0 and calls = 0 and opponents < 4 raise force
```

The fourth line there about folding for any raise covers the rest of the positions. After the blinds the order of those lines doesn't matter because there are not any further conflicts. By the way, did you pick up on the fact that this code will have the bot raise with A9 when folded to in the cutoff, button, or in the small blind? If so, good for you – you are starting to get it.

So always code suited hands before you code their unsuited counterparts, and always code positional exceptions before the general rule for the rest of the positions. (Similarly, post-flop always code made hands before coding draws because from our bot's perspective a made flush or a straight is still a flush draw or a straight draw. More on that in the post-flop section.)

Note: AKs means AK of spades, not AK suited. AK Suited is, you guessed it: AK suited

Card rank values and card suit values can be used on any betting round, not just pre-flop. Often using them to define a post-flop condition is a good way to randomize actions. For example on the river with a possible straight and possible flush out, either of which would require the use of two cards, facing a medium-size bet with two live opponents you might decide to call if you are holding top pair and your hand contains an ace or a queen, or perhaps if your hand is a certain suit. That way you make different decisions for the same situation randomly, perhaps making what is kind of a tough call 50% of the time and folding 50% of the time (more about this below).

3.3.11 Figuring Out Your Position

You will need to use the variables to figure out your position, other than when you are in one of the three positions that there is a condition defined for (*In BigBlind, In SmallBlind, In Button*). This is done by using a combination of the call/fold/raise count and the opponent count variables.

This does require a little thinking, but the variables are there for you to code the bots' position correctly. Many of the code examples given so far have done this.

Keep in mind that the bot does not record its own action in the log. This is important when you are creating instructions for dealing with raises and re-raises after you act. For example if you want to call one re-raise only after you raise pre-flop, this would be a valid way to do that:

```
When (hand = JT suited or hand = QJ suited or hand = KJ suited or hand = AT suited or hand = AJ suited or hand = KQ suited)
   when raises = 1 and BotsLastAction = raise call force
```

Note that the raise count will only be 1 in the bots' log, because it only records opponent's actions not it's own. It uses the raise count in it's log to assess the situation. So you need to get used to that concept when creating code for dealing with actions after the bot acts on the same betting round.

Here is another example of figuring out pre-flop position. This will be for hands that you want to raise with from the cutoff seat or button when folded to, and then fold for a re-raise:

```
When (hand = A suited or hand = A9 or hand = K9 suited or hand = Q9 suited)
  When In SmallBlind and raises = 0 call force
  When raises = 0 and calls = 0 and StillToAct <=3 raise force
  When raises = 1 fold force
```

This code combines several of the elements we have just discussed. By coding the small blind exception first you can make sure it only calls with these hands from that position even when folded to (if you insist on playing that weak). By stating calls = 0 and raises = 0 you know that nobody else has entered the pot yet – it's either been folded to you or so many people are sitting out that you are first to act from the cutoff. The *StillToAct* variable gives you the number players behind you that have not acted yet, and with 3 or less you can only be in the cutoff, button, or small blind.

What if you don't want to play these hands the same way in a short-handed pot, such as a table with only a few players or maybe a full ring game with most players sitting out? In other words you want to make sure there have been some folds in front of you. That is easily done by simply adding a fold count to the code, like this:

```
When (hand = A suited or hand = A9 or hand = K9 suited or had = Q9 suited)
  When in SmallBlind and raises = 0 and folds >=3 call force
  When raises = 0 and calls = 0 and folds >=2 and StillToAct <=3 raise force
  When raises = 1 fold force
```

3.3.12 Randomizing Actions

Because each individual suit can be specified in your hole cards, you can randomize actions for certain situations by cards and card suits. That way you are not playing the same way in the same situation all the time. The classic use for this is in bluffing situations, but there are many borderline calling situations where it's a good idea to call a certain percentage of the time.

Many combinations can be used specifying suited hands, certain suited hands, and specific cards to come up with many different randomization percentages. This can be done within a certain hand or hand group or for an entire situation.

For example on the river with top pair second best kicker you might decide to only call a large bet 25% of the time by specifying that your first hole card is a certain suit, or 50% of the time by specifying two suits that it can be. Preflop you might decide to re-raise from the big blind in short-handed situations 50% of the time with certain holdings by specifying suits, like this:

```
When (hand = Ks J or hand = Kh J or hand = Qs J or hand = Qh J or hand = As T or hand = Ah T)
  When in BigBlind and raises = 1 and OpponentsAtTable <=3 raise force
```

So now at short-handed tables (with three or less opponents sitting in) you will play back aggressively half the time when you are in the big blind facing a raise and holding KJ, QJ, or AT.

Random Variable

There is also a *random* variable that can be used, which is a number chosen to represent a percentage of time that you want to take a certain action. This numeric-valued variable is simply written as: *Random*. It returns a random number between 1 and 100, inclusive. It works like this:

```
When Random <= 75    // true 75% of the time
When Random > 75     // true 25% of the time
When Random = 75     // true 1% of the time
When Random <= 100     // always true
```

Here is an example of it's use:

```
Flop
When bets = 0 and raises = 0 and position = first and opponents <= 3 and not (flushpossible or board = KK or board = QQ or board =
JJ or board = TT) and haveoverpair and random <= 35 call force
```

That code will check 35% of the time you flop an overpair in first position against 3 or less opponents when a flush is not possible and a large pair is not on board. You could then code a line to check-raise with this hand (if somebody bets) like this:

```
When bets = 1 and raises = 0 and position = first and betsize <= 12% stacksize and not (flushpossible or board = KK or board = QQ or
board = JJ or board = TT) and haveoverpair raisepot force
```

By using the <= connector you can just specify what percent of the time you want to take a certain action in the given situation with the number you use. This is the simplest way to code randomized actions.

Randomizing some of your play is critical for successful short-handed play in my opinion, and it's why our bot does this by default with certain hands.

3.3.13 Comments

Comments can be inserted into the code by using a double forward slash // before the comment and the rest of the line will be ignored by the bot. Comments are generally used to give an explanation of the code you are writing. They can also be used to temporarily remove code from your profile.

Here is an example. Comments are shown in green:

```
When hand = 7d 2 suited  // start a new hand group. This is a when condition without an action hence statements below are indented
   when calls = 0 and raises = 0 and folds >= 4 raise 2 force
   //because I want to mix up my game
   when calls < 2 and folds  >= 4 and raises = 1 RaisePot force
   //because I want to play like a total donk and make them laugh when I show seven-deuce
```

Here is another example:

```
When hand = 7d 2 suited  // start a new hand group. This is a when condition without an action hence statements below are indented
   when calls = 0 and raises = 0 and folds >= 4 raise 2 force
   // because I want to mix up my game
   // when calls < 2 and folds >= 4 and raises = 1 RaisePot
   // I no longer want to play like a total donk and so have commented out the above line. I have not deleted it in case I want to play
   // like a donk again in the future. In that case I will just uncomment that line.
```

Please note that line breaks do matter when it comes to comments. Any time you start a line with // that entire line will be ignored, and if the comment carries over to another line like in the above example the beginning of that continuation line must also start with the // comment queue.

3.3.14 Customizable Time Delay

Shanky Technologies poker bots have a built-in randomized time delay of 1 to 3 seconds before acting. We feel this is appropriate for most situations, looks normal, and doesn't hold up the game. However, there may be certain situations when delaying your action further is desirable. For example, when you hold a very strong hand and want to appear as though you are deliberating before acting. In this case, you can add the word *delay* followed by a space and then a number at the very end of your codeline (after your action is specified). This will cause the bot to wait an additional amount of time, specified in seconds by your number, before acting. Here is a preflop example:

```
When in bigblind and raises = 1 and calls = 0 and lastraiserposition <= 4 and hand = AK call force delay 5
```

In this situation you are in the big blind holding AK and face a raise from a player who open-raised in late position. This codeline forces the bot to wait an extra 5 seconds before calling, which might lead your opponent to believe you had to think about whether or not to call, and therefore do not hold as strong of a hand as you actually have. Here is another example from the flop:

```
When haveset and opponents = 1 and bets = 1 and raises = 0 and betsize <= 10 call force delay 7
```

Here you flopped a set and are heads up on the flop in last position against one opponent who bets into you. This codeline forces the bot to wait an additional 7 seconds before calling, which might result in the opponent concluding that your hand is weak and inspiring them to make another bet on the Turn.

Please note the added time delay is in **addition** to the existing random 1-3 second built-in time delay. In some games (especially speed-poker type structures) waiting too long before acting could actually cause you to time out, fold the hand, and be seated out. So don't go overboard and use this feature wisely.

3.4 Preflop Coding

Preflop PPL code must begin with the single word "preflop" all by itself on a single line. It is logical to start with preflop custom coding so the beginning of your custom profile code should look like this:

```
custom
preflop
```

Then you can start making statements which define conditions and give specific actions for specific situations.

3.4.1 Preflop Actions

Beep - *computer will beep at you but take no action on the hand – if you do not respond you will time out*

Call – *check if possible else call*

Play - *same exact action as 'call' above, use either*

Raise - *Generic Raise action. The size of the raise depends on available buttons for the particular poker room, whether we are acting preflop or on the flop turn or river and certain option settings. Will raise minimum preflop unless the "Make Pot Sized Raises for" option is set in which case it will honor those option settings.*

RaiseMin - *will raise minimum*

RaiseHalfPot - *will raise pot or max if half pot button not available and raise min if no other button available*

RaisePot - *will raise max if pot button not available and raise min if no other button available*

RaiseMax - *will raise pot if max button not available and raise min if no other button available*

Fold - *will check if available else fold*

Notice how *Bet* is not included as a preflop action. That's because this action is not an option on the first betting round in Texas Hold'em. That is, before the flop you can only call, fold, or raise due to the blind structure of the game. So don't make the beginner mistake of trying to use *BetPot* or *BetMax* in the preflop section of your code because that will invalidate it. (Use *RaisePot* or *RaiseMax* instead.)

Those actions combined with the list of variables in Appendix 1 give you all the preflop elements. With everything you have read so far, you can completely control how the bot plays before the flop in all different types of Holdem or Omaha games. It is possible to create many different profiles for many different game conditions and store them all in your Bot components folder, loading them as needed.

In fact, because of the *OpponentsAtTable* variable, there really isn't any need for the auto-load profile feature on the bot any more. You can simply create custom codes that play more aggressively when short-handed (and even more aggressively when heads up).

However it's a good idea to program different profiles for short-handed tournament situations than cash games. For example when down to the last few players in an SNG you might simply want to *Fold or Go All In,* but short-handed in cash games of course you would not do that. The advantage to using the custom code over the auto-load profile feature is that you can automatically tighten up when a cash game fills back up, whereas with the auto-load profile feature only one switch can be made durin a session.

Perhaps the best way to really learn the customizing code is to see working examples. Therefore I recommend studying the next section to see how the different code elements fit together.

3.5 Preflop Custom Profile Code Examples

We will start with the simpler custom codes and work our way into the more detailed complete profiles, making sure to use all the pre-flop variables, values, and actions.

3.5.1 Eliminating Hands Preflop

By default the bot will play aggressively when folded to in late position; raising from the cutoff and button with a pretty wide range including J8, J7, K7, and Q7, among others. Let's say you noticed it doing that with some hands that you want to tighten up on. To specify hands you simply never want to play, use this line of code:

```
When (hand = J8 or hand = J7 or hand = Q7) fold force
```

To eliminate hands just from the cutoff seat you can use this code:

```
When (hand = K7 or hand = 22 or hand = 33 or hand = 44) and StillToAct >= 3 fold force
```

(Of course with either code you would adjust for the specific hands you want to fold.)

3.5.2 Trapping in Early Position

The bot by default doesn't slow-play AA or KK. Many players like to trap with them in early position. To do that you can use this simple line of code:

```
When (hand = AA or hand = KK) and raises = 0 and StillToAct >= 7 call force
```

The bot will now slow-play these hands early in unraised pots. If it gets raised behind you the bot will behave as you have your options set when the action comes back to you (raise pot, etc).

3.5.3 Aggressive Play for Tournaments

If you are familiar with the Holdem Bot then you know that the *Aggressive Tournament Setting* on the bot's option menu is a Bet Max function that is set by hand strength. But you might not like the way the hand groups are listed. For example, the first position setting combines AA, KK, and AKs (which of course means AK suited on the option menu). When this setting is invoked the bot will raise All-In only after an opponent raises while holding either AA, KK, or AK suited.

Many players have expressed an interest in dropping AKs from that hand group to use in cash games, which is easily done with this line of code:

```
When (hand = AA or hand = KK) and raises >= 1 RaiseMax force
```

You would then leave the Aggressive Tournament Setting off of course. Now, that line of code makes a great compliment to the Trapping in Early Position code above. To use them both you could combine them in a cleaner piece of code using *when conditions* that looks like this:

```
When (hand = AA or hand = KK)
   When raises = 0 and StillToAct >= 7 call force
   When raises >= 1 RaiseMax force
```

A lot of players will be satisfied with the types of small adjustments we have covered so far – that is, eliminating certain hands from certain positions, trapping with AA+KK early, and choosing their own hands and situations for going all-in preflop. Especially those who are intent upon grinding out rake back or clearing bonuses in cash games. To put all this together in a working custom profile it will look like this:

```
custom
preflop

When (hand = J8 or hand = J7 or hand = Q7) fold force

When (hand = K7 or hand = 22 or hand = 33 or hand = 44) and StillToAct >= 3 fold force

When (hand = AA or hand = KK)
   When raises = 0 and StillToAct >= 7 call force
   When raises >= 1 RaiseMax force
```

That's all there is to it. Adjust the hands and values as you see fit and post the code under your option settings in your saved profile and you are good to go.

3.5.4 Programming Fold or All-in Profiles

These are easiest custom profiles to create because you are only concerned with either folding or going all-in before the flop. This is a popular way to play SNG's. The simplest type of code wouldn't even bother with position or the opponent count and simply shove or fold by hand strength and perhaps take into account if there has already been a raise or not. (This approach would probably be more adept for Turbo SNG's.) So let's start there.

```
custom
preflop

When (hand = AA or hand = KK or hand = QQ or hand = JJ or hand = TT or hand = 99 or hand = AK or hand = AQ suited)
RaiseMax force
```

```
When (hand = A suited or hand =KQ suited or hand = AQ or hand = AJ or hand = 88 or hand = 77 or hand = 66 or hand = 55)
   and raises = 0 RaiseMax force

When (hand = AT or hand =A9 or hand = A8 or hand = A7 or hand = 44 or hand = 33 or hand = 22 or hand = KQ)
   and raises = 0 and calls <= 1 RaiseMax force

When (hand = A or hand =QJ suited or hand = JT suited or hand = T9 suited or hand = 98 suited or hand = 87 suited or hand = 76
suited or hand = KJ suited)
   and raises = 0 and calls <= 1 and StillToAct <= 3 RaiseMax force

When (hand = K suited or hand = KJ or hand = KT or hand = QJ or hand = QT or hand = JT or hand = T9)
   and raises = 0 and calls = 0 and OpponentsAtTable = 1 RaiseMax force

When others
   When others fold
```

There you have a decent Turbo SNG template that you can do what you want with; adjust the hands as you like etc. Notice that there is a different grouping of hands for situations that require no raises in front of you, no raises in front of you plus no more than 1 caller, no raises plus no more than one caller plus no more then three opponents behind you, and finally a hand group for when you are heads up and first to act.

This is a complete profile as it excludes every other hand by using the *when others fold* command at the end. So it doesn't matter what your preflop option settings are because they will be ignored. Of course when the bot gets a free look at the flop it will still play according to your post-flop option settings (for a profile such as this I would have the post-flop side of the Aggressive Tournament setting set to second position).

For non-turbo SNG's you are probably going to want a more reasonable playing profile in the beginning, much as you would at the beginning of an MTT or even a cash game type setting. But as the game winds down if you are still there you would likely want to start getting aggressive and perhaps even start playing the above profile.

For example you might be perfectly content to play according to your option settings until it gets short-handed and then have the above profile kick in. This can easily be accomplished by removing the *when others* command at the bottom and inserting the *OpponentsAtTable* variable requirement for all your statements.

3.5.5 Taking this one step farther

We can combine our earlier code where we set up trapping for early positions and eliminated some of the blind-stealing hands we didn't want into the whole thing and end up with this code:

```
custom
preflop

When (hand = J8 or hand = J7 or hand = Q7) fold force

When (hand = K7 or hand = 22 or hand = 33 or hand = 44) and StillToAct >= 3 fold force

When (hand = AA or hand = KK)
  When raises = 0 and StillToAct >= 7 call force
  When raises >= 1 RaiseMax force

When (hand = AA or hand = KK or hand = QQ or hand = JJ or hand = TT or hand = 99 or hand = AK or hand = AQ suited)
   When OpponentsAtTable <= 4 RaiseMax force

When (hand = A suited or hand =KQ suited or hand = AQ or hand = AJ or hand = 88 or hand = 77 or hand = 66 or hand = 55)
   When OpponentsAtTable <= 3 and raises = 0 RaiseMax force

When (hand = AT or hand =A9 or hand = A8 or hand = A7 or hand = 44 or hand = 33 or hand = 22 or hand = KQ)
   When OpponentsAtTable <= 2 and raises = 0 and calls <= 1 RaiseMax force

When (hand = A or hand =QJ suited or hand = JT suited or hand = T9 suited or hand = 98 suited or hand = 87 suited or hand = 76
suited or hand = KJ suited)
   When OpponentsAtTable <= 2 and raises = 0 and calls = 0 RaiseMax force

When (hand = K suited or hand = KJ or hand = KT or hand = QJ or hand = QT or hand = JT or hand = T9)
   and raises = 0 and calls = 0 and OpponentsAtTable = 1 RaiseMax force
```

If you are still a little confused about this you should study the above code and see how it changed from the previous one. Aside from combining our earlier code, all the statements now have an *OpponentsAtTable* requirement and that requirement keeps getting smaller (wanting less players) as we add the weaker hand groups. That way the bot starts pushing more as the game gets more and more short-handed. Note that the hand group which starts with *hand = A* (any ace) no longer needs the *StillToAct* variable as there cannot be more than 3 opponents StillToAct when a maximum of two are sitting at the table in the first place – so it has been removed.

It's also important to understand that because we used the <= comparator, each statement includes the situations underneath it as well. This is because we started coding for more opponents and worked our way into less and less opponents. That way the bot still raises max with the better hands even when you get down to heads up play, without having to restate those actions. It's just the weaker hand groups that get added as the player count is reduced.

We also left off the *when others* + *when others* fold command at the end, allowing the bot to play a reasonable game from its default programming for other playable situations that are not covered. The result is a very usable 1-table SNG code, especially for the Turbo-style games.

3.5.6 Programming for Hand Groups

Now we are getting into the building blocks for designing an entire playing profile. As you have already learned, using multiple *when conditions* in a statement is the best way to code all possible actions for a hand or group of hands. You can do this just for certain hands that you want played differently than you can set them on the option menu, or you can design an entire playing profile hand by hand.

Here is an example of creating a code for playing middle suited connectors in Limit games.

```
When (hand = T9 suited or hand = 98 suited or hand = 87 suited)
   When BotsLastAction = raise call force
   When (calls = 1 or calls = 2) and raises = 0 call force
   When calls <= 2 and raises >=1 fold force
```

```
    When calls = 3 and raises <= 1 call force
    When calls = 3 and raises > 1 fold force
    When calls >=4 call force
    When calls = 0 and raises = 0 and StillToAct <= 3 raise force
    When calls = 0 and raises = 0 and StillToAct > 3 call force
```

This would only be of use in Limit games because there is no BetSize variable used and the bot would therefore call raises of any size in NL games – which is a good formula for disaster. What you should notice about this code is that **it covers all possible call/fold/raise scenarios**. That is what you should do when creating custom code for hand groups, so you can be assured the bot will always play the way you want it to. If you let a scenario slip through the cracks the bot might fold hands you want to play or vice-versa.

So you need to always look at your code and think about making sure you covered all the possible raise/fold/call counts. A good way to do this is choose either the fold, call, or raise count and start every line in the statement with that, as I did above by using the call count. This will make it much easier to scan and make sure there are no holes in it, so to speak.

However you can use the *when others* condition to catch everything you don't want to have to code for as long as the action is the same for all of them. For example in the above code the last line could be replaced with:

```
When others call force
```

Did you also notice that the second line (in the above code for suited connectors) is an instruction on how to handle re-raises behind it? This goes first because code is read in sequential order and I want to bot to call any and all subsequent re-raises those times that it puts in the first raise. If I put that line last it wouldn't get executed because code is read in sequential order (and the first match is executed). I know this because I am only raising into 3 or less *StillToAct* opponents and will only call a raise with 3 or more callers in the pot – so there couldn't be enough callers if somebody re-raised me. By putting this line first it handles the exception for raises behind me that I want. Remember to always put exceptions at the beginning of your statements. (Note that I don't want any blind exceptions for this hand group).

Now let's change that piece of code for a NL game and use the *when others* catchall to finish it.

```
When (hand = T9 suited or hand = 98 suited or hand = 87 suited)
    When BotsLastAction = raise and BetSize <= 8 call force
    When (calls = 1 or calls = 2) and raises = 0 call force
    When calls <= 2 and raises =1 and BetSize > 4 fold force
    When calls <= 2 and raises =1 and BetSize < 4 call force
    When calls >= 3 and raises = 1 and BetSize < 6 call force
    When calls >= 3 and raises = 1 and BetSize >= 6 fold force
    When raises > 1 fold force
    When calls = 0 and raises = 0 and StillToAct <= 3 raise force
    When others call force
```

Here I have plugged in how big of a raise I am willing to call preflop (also based on the number callers), and it's good to go for NL/PL games. Note that I was able to change >= into simply = for 1 raise being that I am just going to fold for more than one raise, as that will always be more than I am willing to call.

3.5.7 Using the StackSize Variable for NL Games

There are several ways to incorporate stack size recognition into your preflop custom profile using the *StackSize* variable. One is to simply assign it a numerical value for making action decisions, probably calling or raising actions when your stack is getting low. Here is an example of that:

```
When StackSize <= 20
   When (hand = AA or hand = KK or hand = QQ or hand = JJ or hand = TT or hand = 99 or hand = 88 or hand = 77 or hand = AK or
hand = AQ or hand = AJ suited or hand = KQ suited) RaiseMax force

When StackSize <= 15
   When (hand = 66 or hand = 55 or hand = 44 or hand = 33 or hand = 22 or hand = AJ or hand = AT or hand = A9 suited or hand =
KQ) RaiseMax force

When StackSize <= 10
   When (hand = A suited or hand = KJ suited or hand = QJ suited or hand = JT suited or hand = T9 suited or hand = 98 suited)
   RaiseMax force

When StackSize <= 7
   When (hand = A or hand = K suited or hand = 87 suited or hand = 76 suited) RaiseMax force
```

Remember, the numerical value for *StackSize* is the number of big blinds. This code defines which hands to push with as your stack gets shorter and shorter and is of tremendous use in all tournament situations. You should put this at the top of your entire custom preflop code as it is an exception to everything and you want the bot to find it first when you are getting short-stacked.

3.5.8 Using % Comparators

Another way to use *StackSize* in your custom preflop code is with the **Relative Potsize/Stacksize Comparison Condition** (explained in point 3 of section 3.2.1 of this manual). This method involves using the BetSize, AmounttoCall, TotalInvested, StartingStackSize, or any of the opponent stack size variables and then a comparator symbol to the StackSize variable. For example:

```
When AmounttoCall  < 33% StackSize call force
```

In this case the bot will call when an opponent made a bet or raise which requires less than a third of your stack size to call. You would, of course, usually want to further define the situation.

For making sure you never fold when you are pot-committed, you can use the *TotalInvested* variable to define the point where you have put so much of your stack in the pot that you simply do not want to fold. This is a safe code to use in just about any NL game situation; because you know that you won't be putting most of your stack in the pot without a decent holding anyway. Here are some examples:

```
When TotalInvested  > 100% StackSize call force   //will never fold once you invest half your stack

When TotalInvested  >= 120% StackSize call force   //will never fold once you invest most of your stack

When TotalInvested  >= 200% StackSize call force   //more conservative, requires 2/3 of your stack invested before defending
```

That type of code is very important for tournament profiles, by the way. You don't want to fold with just a few chips left after committing most of your stack. If you are not coding a complete profile you can skip this step and just use the option menu setting on our bot to make sure you never fold when pot-committed.

If you are coding a complete NL profile and excluding all option menu settings and default programming, then you need to decide where that pot-committed threshold is and insert this type of code in a strategic place on all your betting rounds. But be careful you don't stop your bot from betting and raising when you want, as the above lines of code instruct it to just call when the condition is true (which means checking and calling). To prevent that you could use this type of code instead:

```
When TotalInvested > 120% StackSize and (bets >=1 or raises >=1) RaiseMax force
```

Conversely, in the late stages of a tournament you might want to push against a short-stacked opponent with any two cards when your stack is so big it cannot be damaged:

```
When MaxCurrentOpponnetStacksize < 8% stacksize raisemax force
```

List of % Comparators

There are eleven variables that can be used with a % comparison, and of those nine can only go on the left of the percent sign. The other two must always be on the right. They are:

Left Side	%	Right Side
BetSize		PotSize
TotalInvested		StackSize
AmountToCall		
StartingStackSize		
MaxCurrentOpponentStackSize		
MaxOpponentStackSize		
MinOpponentStackSize		
MaxStillToActStackSize		
MinStillToActStackSize		

In addition, this special situation % comparator is allowed:

Left Side	%	Right Side
PotSize		Stacksize

Please note that Potsize can only go on the left when Stacksize is on the right.

Therefore the 19 *comparisons* allowed are:

Betsize % PotSize
Betsize % StackSize
TotalInvested % PotSize
TotalInvested % StackSize
AmountToCall % PotSize
AmountToCall % StackSize
StartingStackSize % PotSize (of questionable use)
StartingStackSize % StackSize
MaxOpponentStackSize % PotSize
MaxOpponentStackSize % StackSize
MaxOpponentStackSize % PotSize
MaxOpponentStackSize % StackSize
MinOpponentStackSize % PotSize
MinOpponentStackSize % StackSize
Potsize % StackSize
MaxStilltoAct % PotSize
MaxStilltoAct % StackSize
MinStilltoAct % PotSize
MinStilltoAct % StackSize

The above is currently the only structure supported when using the % comparators.

You can also use this method to call raises more liberally when your stack size is big. For example:

```
When (hand = 66 or hand = 55 or hand = 44 or hand = 33 or hand = 22)
   When AmounttoCall  < 2% StackSize call
```

Conversely, you can protect a big stack in tournaments by tightening up preflop. For example:

```
When StackSize > 75 and Betsize > 8 and not (hand = AA or hand = KK or hand = QQ or hand = AK or hand = KQ suited) fold force
```

(That last line of code introduced using the *not* operator. This can be a bit tricky so there is a detailed discussion of it in the post-flop coding section below.)

In situations where it may be important that your **effective stack size** is less than your stack, you can say:

```
When havestraightdraw and not (havestraight or haveflush or havepair)
   when MaxCurrentOpponentStackSize < 40% stacksize fold force
```

The above code would keep you from drawing to a straight when the only stack you can win is small.

3.5.9 Example of a Complete Preflop Fixed Limit Profile

Below is an example of a complete preflop custom profile for playing Fixed Limit games. It has been debugged for typos (which we will show you how to do below) and will work if you want to give it a whirl. You should easily be able to use it as a template and alter it the way you want.

```
custom
preflop

When (hand = AA or hand = KK or hand = AK suited) raise force

When (hand = AK or hand = QQ)
  When raises <= 1 raise force
  When (BotsLastAction = raise or BotsLastAction = call) call force
  When raises > 1 call force

When (hand = AQ suited or hand = AJ suited)
  When raises = 0 raise force
  When (BotsLastAction = raise or BotsLastAction = call) call force
  When raises = 1 call force
  When raises = 2 and calls >= 3 call force
  When raises = 2 and calls < 3 fold force
  When raises >2 fold force

When (hand = KQ suited or hand = JJ or hand = TT)
  When calls >= 4 call force
  When StillToAct >= 6 and raises = 0 call force
  When (BotsLastAction = raise or BotsLastAction = call) call force
  When raises = 0 raise force
  When raises = 1 call force
  When raises = 2 and calls >= 3 call force
  When raises = 2 and calls < 3 fold force
  When raises >2 fold force

When hand = AQ
  When BotsLastAction = raise call force
  When calls >= 5 and raises = 0 call force
  When raises = 0 raise force
  When raises = 1 call force
  When raises > 1 fold force

When (hand = AT suited or hand = KJ suited)
  When (BotsLastAction = raise or BotsLastAction = call) call force
  When raises = 0 and calls <= 1 and StillToAct <= 4 raise force
  When raises = 1 and calls >= 3 call force
  When raises = 1 and calls < 3 fold force
  When raises > 1 fold force
  When others call force

When (hand = KT suited or hand = QT suited)
  When BotsLastAction = raise call force
  When BotsLastAction = call and calls >= 3 call force
  When raises = 0 and calls = 0 and StillToAct <= 3 raise force
  When raises = 1 and calls >= 3 call force
  When raises = 1 and calls < 3 fold force
  When raises > 1 fold force
  When others call force

When (hand = QdJd or hand = Js Ts or hand = Tc9c or hand = 9h8h)
  When (In BigBlind or In SmallBlind or In Button) and raises = 0 and calls >= 4 raise force

When (hand = QJ suited or hand = JT suited or hand = T9 suited or hand = 98 suited)
  When (BotsLastAction = raise or BotsLastAction = call) call force
  When raises = 0 and calls = 0 and StillToAct <= 3 raise force
  When others call force

When (hand = AJ or hand = KQ)
  When BotsLastAction = raise call force
  When in BigBlind and raises <= 1 call force
  When raises = 0 and StillToAct >=5 call force
  When raises = 0 and opponents < 5 raise force
  When raises = 1 and calls >= 2 fold force
  When raises = 1 and calls < 2 call force
  When raises > 1 fold force
  When others call force

When (hand = 99 or hand = 88 or hand = 77)
  When BotsLastAction = raise call force
  When BotsLastAction = call and calls >= 3 call force
  When In BigBlind and raises = 1 call force
  When raises = 0 and calls = 0 and StillToAct <= 4 raise force
  When raises = 1 and calls >= 3 call force
  When raises = 1 and calls < 3 fold force
  When raises > 1 fold force
  When others call force

When (hand = AT or hand = KJ or hand = QJ)
  When BotsLastAction = raise call force
  When In BigBlind and raises = 1 call force
  When raises >= 1 fold force
  When raises = 0 and calls = 0 and opponents <= 4 raise force
  When others call force

When (hand = KT or hand = QT or hand = JT)
  When BotsLastAction = raise call force
  When In BigBlind and raises = 1 call force
  When raises >= 1 fold force
  When raises = 0 and calls = 0 and opponents <= 3 raise force
  When StillToAct >= 7 fold force
  When others call force

When (hand = 6d6c or hand = 5s5h or hand = 4d4s)
  When (In BigBlind or In SmallBlind or In Button) and calls >= 4 raise force

When (hand = 66 or hand = 55 or hand = 44 or hand = 33 or hand = 22)
```

```
  When BotsLastAction = raise call force
  When In BigBlind and raises = 1 call force
  When In SmallBlind and raises = 0 and calls >= 1 call force
  When raises = 0 and calls = 0 and StillToAct <= 2 raise force
  When raises = 0 and calls >= 2 call force
  When others fold force

When (hand = A9 suited or hand = A8 suited or hand = A7 suited)
  When (BotsLastAction = raise or BotsLastAction = call) call force
  When In BigBlind and raises = 1 call force
  When raises = 0 and calls = 0 and StillToAct <= 4 raise force
  When raises >= 1 fold force
  When StillToAct >= 7 fold force
  When others call force

When (hand = A6 suited or hand = A5 suited or hand = A4 suited or hand = A3 suited or hand = A2 suited)
  When BotsLastAction = raise call force
  When BotsLastAction = call and calls >= 3 call force
  When In BigBlind and raises = 1 call force
  When In SmallBlind and raises = 0 and calls >= 1 call force
  When raises = 0 and calls = 0 and StillToAct <= 2 raise force
  When raises >= 1 fold force
  When raises = 0 and calls >= 3 call force
  When others fold force

When Hand = A9
  When BotsLastAction = raise call force
  When In BigBlind and OpponentsAtTable <= 4 and raises = 1 and calls = 0 raise force
  When In BigBlind and raises = 1 call force
  When In SmallBlind and raises = 0 and calls >= 1 call force
  When raises = 0 and calls = 0 and opponents <= 3 raise force
  When others fold force

When (Hand = A8 or hand = A7)
  When In BigBlind and raises = 1 call force
  When In SmallBlind and raises = 0 and calls >= 1 call force
  When raises = 0 and calls = 0 and opponents <= 3 raise force
  When others fold force

When (Hand = A6 or hand = A5)
  When In BigBlind and raises = 1 call force
  When In SmallBlind and raises = 0 and calls >= 1 call force
  When raises = 0 and calls = 0 and opponents <= 2 raise force
  When others fold force

When (hand = A4 or hand = A3 or hand = A2)
  When In SmallBlind and raises = 0 and calls >= 1 call force
  When In BigBlind and OpponentsAtTable <= 4 and raises = 1 and calls = 0 call force
  When raises = 0 and calls = 0 and opponents <= 2 raise force
  When others fold force

When (Hand = Ks9s or hand = Kh9h or hand = Ks8s or hand = Kd8d or hand = Kc7c or hand = Kd7d)
  When BotsLastAction = raise call force
  When BotsLastAction = call and calls >= 3 call force
  When In SmallBlind and raises = 0 and calls >= 1 call force
  When In BigBlind and OpponentsAtTable <= 4 and raises = 1 and calls = 0 raise force
  When In BigBlind and raises = 1 call force
  When raises = 0 and calls = 0 and opponents <= 3 raise force
  When raises = 0 and calls >=3 call force
  When calls >= 4 call force
  When others fold force

When (Hand = K9 suited or hand = K8 suited or hand = K7 suited)
  When BotsLastAction = raise call force
  When In SmallBlind and raises = 0 and calls >= 1 call force
  When In BigBlind and raises = 1 call force
  When raises = 0 and calls = 0 and opponents <= 2 raise force
  When raises = 0 and calls >=4 call force
  When others fold force

When (Hand = Ks6s or hand = Kh6h or hand = Ks5s or hand = Kd5d or hand = Kc4c or hand = Kd4d)
  When BotsLastAction = raise call force
  When BotsLastAction = call and calls >= 4 call force
  When In SmallBlind and raises = 0 and calls >= 1 call force
  When In BigBlind and OpponentsAtTable <= 4 and raises = 1 and calls = 0 call force
  When raises = 0 and calls = 0 and opponents <= 2 raise force
  When raises = 0 and calls >=4 call force
  When others fold force

When (Hand = K6 suited or hand = K5 suited or hand = K4 suited or hand = K3 suited or hand = K2 suited)
  When BotsLastAction = call and calls >= 4 call force
  When In SmallBlind and raises = 0 and calls >= 1 call force
  When raises = 0 and calls >=4 call force
  When others fold force

When Hand = K9
  When BotsLastAction = raise call force
  When In SmallBlind and raises = 0 and calls >= 1 call force
  When In BigBlind and OpponentsAtTable <= 4 and raises = 1 and calls = 0 call force
  When raises = 0 and calls = 0 and opponents <= 3 raise force
  When others fold force

When (hand = K8 or hand = K7)
  When BotsLastAction = raise call force
  When In SmallBlind and raises = 0 and calls >= 1 call force
  When In BigBlind and OpponentsAtTable <= 3 and raises = 1 and calls = 0 call force
  When raises = 0 and calls = 0 and opponents <= 2 raise force
  When others fold force

When (Hand = Qd9d or hand = Qs9s or hand = Qd8d or hand = Qs8s)
  When BotsLastAction = raise call force
  When BotsLastAction = call and calls >= 4 call force
  When In SmallBlind and raises = 0 and calls >= 1 call force
  When In BigBlind and OpponentsAtTable <= 3 and raises = 1 and calls = 0 raise force
  When In BigBlind and raises = 1 call force
  When raises = 0 and calls = 0 and opponents <= 2 raise force
  When raises = 0 and opponents = 3 call force
  When raises = 0 and calls >=3 call force
  When others fold force

When (Hand = Q9 suited or hand = Q8 suited or hand = Q7 suited or hand = J9 suited)
  When BotsLastAction = raise call force
  When BotsLastAction = call and calls >= 4 call force
  When In SmallBlind and raises = 0 and calls >= 1 call force
  When In BigBlind and raises = 1 call force
  When raises = 0 and calls = 0 and opponents <= 2 raise force
  When raises = 0 and calls >=3 call force
  When others fold force

When (Hand = Q6 suited or hand = Q5 suited or hand = Q4 suited or hand = Q3 suited or hand = Q2 suited)
  When BotsLastAction = raise call force
  When BotsLastAction = call and calls >= 5 call force
  When In SmallBlind and raises = 0 and calls >= 1 call force
  When In BigBlind and raises = 1 call force
  When raises = 0 and calls = 0 and opponents = 1 raise force
  When raises = 0 and calls >=4 call force
  When others fold force

When (Hand = 87 suited or hand = 76 suited)
  When BotsLastAction = raise call force
  When BotsLastAction = call and calls >= 3 call force
  When In SmallBlind and raises = 0 and calls >= 1 call force
  When In BigBlind and raises = 1 call force
  When raises = 0 and calls = 0 and opponents <= 2 raise force
  When raises = 0 and calls >=2 call force
  When calls >= 4 call force
  When others fold force

When (Hand = 65 suited or hand = 54 suited)
  When BotsLastAction = raise call force
  When BotsLastAction = call and calls >= 3 call force
  When In SmallBlind and raises = 0 and calls >= 1 call force
  When In BigBlind and raises = 1 call force
  When raises = 0 and calls >=3 call force
  When raises = 0 and calls = 0 and OpponentsAtTable = 1 raise force
  When calls >= 4 call force
  When others fold force

When (Hand = J8 suited or hand = J7 suited or hand = T8 suited or hand = T7 suited)
  When BotsLastAction = raise call force
  When BotsLastAction = call and calls >= 5 call force
  When In SmallBlind and raises = 0 and calls >= 1 call force
  When In BigBlind and raises = 1 call force
  When raises = 0 and calls = 0 and opponents <= 2 raise force
  When raises = 0 and calls >= 4 call force
  When others fold force

When (Hand = Q9 or hand = Q8 or hand = J9 or hand = T9 or hand = Q7 or hand = J8 or hand = J7 or hand = 98)
  When BotsLastAction = raise call force
  When In SmallBlind and raises = 0 and calls >= 1 call force
  When In BigBlind and OpponentsAtTable <= 3 and raises = 1 and calls = 0 call force
  When raises = 0 and calls = 0 and opponents <= 2 raise force
  When others fold force

When (Hand = 97 suited or hand = 96 suited or hand = 86 suited or hand = 85 suited or hand = 7s5s or hand = 6h4h)
  When BotsLastAction = raise call force
  When BotsLastAction = call and calls >= 6 call force
  When In SmallBlind and raises = 0 and calls >= 1 call force
  When In BigBlind and OpponentsAtTable <= 3 and raises = 1 and calls = 0 call force
  When raises = 0 and calls >= 4 call force
  When raises = 0 and calls = 0 and OpponentsAtTable = 1 raise force
  When others fold force

When (Hand = T8 or hand = T7 or hand = 97)
  When BotsLastAction = raise call force
  When In SmallBlind and raises = 0 and calls >= 1 call force
  When In BigBlind and OpponentsAtTable <= 2 and raises = 1 and calls = 0 call force
  When raises = 0 and calls = 0 and OpponentsAtTable = 1 raise force
  When others fold force

When (Hand = 75 suited or hand = 64 suited)
  When BotsLastAction = raise call force
  When BotsLastAction = call and calls >= 6 call force
  When In SmallBlind and raises = 0 and calls >= 1 call force
  When In BigBlind and OpponentsAtTable = 1 and raises = 1 call force
  When raises = 0 and calls >= 5 call force
  When raises = 0 and calls = 0 and OpponentsAtTable = 1 raise force
  When others fold force

When (hand = J suited or hand = T suited)
  When BotsLastAction = raise call force
  When BotsLastAction = call and calls >= 6 call force
  When In SmallBlind and raises = 0 and calls >= 1 call force
  When In BigBlind and OpponentsAtTable = 1 and raises = 1 call force
  When raises = 0 and calls >= 6 call force
  When raises = 0 and calls = 0 and OpponentsAtTable = 1 raise force
  When others fold force

When (Hand = 96 or hand = 86 or hand = 76 or hand = 65)
  When BotsLastAction = raise call force
  When In SmallBlind and raises = 0 and calls >= 1 call force
  When In BigBlind and OpponentsAtTable = 1 and raises = 1 call force
  When raises = 0 and calls = 0 and OpponentsAtTable = 1 raise force
  When others fold force

When (hand = 95 or hand = 85 or hand = 75)
  When In SmallBlind and raises = 0 and calls >= 1 call force
  When raises = 0 and calls = 0 and OpponentsAtTable = 1 raise force
  When others fold force

When others
  When others fold force
```

3.6 Post-flop Coding

Post-flop coding is a natural continuation of preflop coding. If you have understood preflop coding above then you only need to make a slight adjustment in your thinking to code for the Flop, Turn and River. On all the subsequent betting rounds coding becomes multi-dimensional, allowing you to incorporate previous round action into your statement conditions.

As you might expect, there are many more variables available for post-flop coding than for preflop. The full list of variables is given in Appendix 1, but we will attempt to address the most prominent ones for our examples in this section.

Post-flop code must begin with the single word *flop*, *turn* or *river* all by itself on a single line depending on which game state you are coding for. Then you can start making statements which define conditions and give specific actions for specific situations. For example:

```
flop

When not (FlushPossible or StraightPossible or PairOnBoard)
   When HaveFlushDraw and HaveStraightDraw Raise force
```

Notice in the above example we first defined a board state, then specified a hand strength and corresponding action to take. This is a very simple piece of code as it doesn’t also define the action at the table, your position, or the number of opponents in the condition. But it shows you a logical order for writing your post-flop code, which follows the same type of order as preflop coding by hand groups.

If you are creating a small code just to adjust for certain situations, you only need to concern yourself with those situations as you will simply let your option settings handle the rest. We really believe that this is the best way to code a custom profile for our poker bots. Just watch the bot play for a few hours, take notes, see what you cannot adjust to your liking from the option settings, and create code snippets as needed.

An example might be if you want to ease up just a bit on the automatic flop-continuation bets when the bot was the last preflop aggressor. You might write a piece of code like this to accomplish your objective:

```
flop

When BotIsLastRaiser and bets = 0 and raises = 0
  When opponents  = 2 and not (FlushPossible or board = A) and not (board = K and StraightPossible) BetPot force
  When opponents  = 1 and (not FlushPossible) BetPot force
```

That code defines all the situations that you want to make the automatic continuation bet on the flop for, so you can now adjust the option setting *Never Bet Flop with Hand Below Middle Pair* to *always* (in fact if you don’t do that this code is pointless, as the bot is more aggressive with flop continuation-betting by default).

Now let’s suppose that you see the bot continuation-betting the flop just the way you want, but decide that you would like to slow-play big hands from early position as well. So now you need to add to your little piece of flop code, and might do it like this:

```
Flop

When BotIsLastRaiser and bets = 0 and raises = 0
  When (HaveFullHouse or HaveStraightFlush or HaveNutFlush or HaveTrips or HaveNutStraight or HaveSet) call force
  When opponents =2 and not (FlushPossible or board = A) and not (board = K and StraightPossible) BetPot force
  When opponents = 1 and (not FlushPossible) BetPot force

When position = first
    When (HaveFullHouse or HaveStraightFlush or HaveNutFlush) call force
    When (HaveTrips or HaveNutStraight or HaveSet or HaveTwoPair) and SuitsOnBoard = 3 and bets = 0 call force
    When HaveTopPair and SuitsOnBoard = 2 BetMin force
```

Remember, there is no action in PPL for *check*. (The only use of that term in the code is as a special value for the unique numerical variable *BotsLastAction*.) Therefore when you want to program a check you must use the action *call*. This is different from preflop coding because before the flop *call* always means putting money in the pot, whereas post-flop when bets = 0 and raises = 0 the action *call* will simply check.

Therefore in order to program a check-raise you need to code for a call when in early position and there are no bets and raises. (If you are not coding a complete profile you don't need to bother coding the subsequent raise with strong hands, as the bot's default programming will do that once the bet count is above zero.) The above code snippet is a good example of this.

Notice how you defined two separate types of conditions there, neither of which bother to first define a board state. The first one simply identified that there were preflop raises and you were the last aggressor, and the second one only concerned yourself with position and holding a very strong hand (*although SuitsOnBoard = 3* does define a board state for one of the actions).

A natural progression for this type of simple "customizing as needed" approach would be to find situations where you want to be more cautious on the Turn and value-bet the River more. After a few days of watching, coding, watching, and coding, you will likely have a profile that performs exactly as you want without having to spend weeks creating 3,000+ lines of code.

On the other hand if you are out to build your own complete profile from scratch, it is probably going to be easiest to organize your post-flop code by board state and then code for all your possible conditions under them (by using the *when condition without an action* approach).

3.6.1 Defining the Board State

Way back in section 3.2.1 we introduced you to **board-specific conditions**, where you can specify cards on the board. We used it in the above example a bit as a reminder. This designation is handy when you want to specify whether or not an ace is on board, or perhaps to randomize your play in certain situations based on certain ranks and/or certain suits being on board. For example:

```
When board = Kc Q7 and not (HavePair or RaisesBeforeFlop) and bets = 1 RaisePot force
```

…as a rare, randomized bluff-raise. I'm sure there are other uses for board-specific conditions as well, but most likely you are going to want to start your post-flop code blocks by using the Boolean variables that were created for identifying specific board states.
They are:

FullHouseOnBoard
QuadsOnBoard
TripsOnBoard
FlushOnBoard
StraightOnBoard
TwoPairOnBoard
PairOnBoard
FlushPossible
StraightFlushPossible
StraightFlushPossibleByOthers
OneCardFlushPossible
StraightPossible
OneCardStraightPossible
OnlyOneStraightPossible
Only1OneCardStraightPossible

Using the *not* operator is necessary for defining most of the more common board states. For example:

```
When not (PairOnBoard or FlushPossible or OneCardStraightPossible)
```

…is a very common board state where you are going to want to bet and raise most of your stronger one-pair hands. However you will also probably want to be leery of big raises from a possible set, straight, or two-pair hand. You can eliminate likely straights by using the board-specific conditions like this:

```
When not (PairOnBoard or FlushPossible or OneCardStraightPossible or board = QJT or board = KJT or board = KQT or board =
AJT or board = AKT or board – AQT or board = KQJ or board = AKQ or board = AQJ or board = AKJ)
```

As you can see most of your opponent's likely straight and two-pair holdings have been eliminated in that condition. This makes it quite a bit safer to re-raise big with an overpair.

3.6.2 Using the *not* Operator

We warned you earlier that there are some pitfalls to watch out for when using *not* to define board states. The main thing you have to be careful of is combining it with the *or* operator in a way that will cause the stated action to take place when any of the conditions is true, when in actuality what you want is for all the conditions to be true when the desired action takes place. Here is an example:

Wrong Way

```
When PairOnBoard and FlushPossible
  when (not HaveFullHouse or not HaveQuads or not HaveFlush) fold force
```

The problem with that piece of code is not obvious, and it has to do with the reverse logic of the *not* operator. What will happen as a result of this code is the bot will fold a full house or even four of a kind if it does not also have a flush! Why? Because you used the *or* operator in conjunction with the *not* operator in a way that told the bot any of those conditions can be true for it to fold.

So you have quads, but the bot reads that if it doesn't have a full house **or** if it doesn't have quads **or** if it doesn't have a flush it should fold – and you don't have a flush so it folds. To put it another way, you specified three possible conditions, any of which being present, should make the bot fold.

You can't combine *or* conditions together when using the *not* operator in the same manner that you do when not using the *not* operator. Because this is easy to get tripped up on, I have developed a red-letter rule that if followed should keep you out of trouble:

When combining *or* conditions while using the *not* operator, enclose the *or* conditions in brackets with a single *not* operator left outside the brackets.

Using the red-letter rule you would rewrite the above incorrect code like this:

Right Way

```
When PairOnBoard and FlushPossible
  When not (HaveFullHouse or HaveQuads or HaveFlush) fold force
```

At first glance this may not look any different from an instructional standpoint than the previous code. However the bot reads this instruction differently and will behave as intended. By bracketing the conditions together **after** using a single *not* operator you are in effect making them like one condition instead of three separate ones. They have been tied together so the action will not be triggered if any one of them are true, whereas in the wrong way example they were three separate conditions any one of which would trigger the action if were true.

Don't worry if this is still confusing – just remember the red-letter rule when using the *not* operator in lines that have multiple *or* conditions defined.

3.6.3 Using the Numerical-valued Variables

Here is a partial list of numerical-valued variables that you can use post-flop:

AmountToCall
Bets
BetSize
BigBlindSize
BotsLastAction (doesn't use numbers, has its own special values)
Calls
CallsSinceLastRaise
Checks
Folds
LowCardsInHand (for Omaha Hi-Lo)
LowCardsOnBoard (for Omaha Hi-Lo)

MaxCurrentOpponentStackSize
NutFullHouseOrFourOfAKind (has special values)
Opponents
OpponentsAtTable
OpponentsLeft
Overcards
OvercardsOnBoard (Overcards to hand)
Position (has special values)
PotSize
Raises
RaisesSinceLastPlay
Random
StackSize
StartingStackSize
StillToAct
SuitsOnBoard
TotalInvested

The complete list is included in Appendix 1, along with explanations of their use and any restrictions. We want to give a few of them additional treatment here to make sure you understand their proper use.

NutFullHouseOrFourOfAKind – be careful with this one, it's tricky. It is so handy for paired board states, however, that it is almost necessary, especially in Omaha games. Remember that it ignores straight-flush possibilities, and that the better hands have a lower number rank. So the nuts is 1, the 2nd nuts is 2, etc.

Therefore when using the < and > symbols to assign values remember that smaller is higher. If you point the arrows in the wrong direction you will end up raising with the lower-valued hands instead of the higher valued ones. You also must account for the = 0 value, because it is lower than the number 1. Understand that 0 simply means you do not have a full house or four of a kind. Therefore you must code for it when using a < symbol - otherwise it will get included in your actions.

This is best explained with examples:

Wrong Way

```
When TripsOnBoard
  When NutFullHouseOrFourOfAKind > = 2 RaisePot force
  When others fold
```

That code may seem intuitively correct because you want to raise with the 2nd nuts or better. But in reality it will raise with the 2nd nuts or worse! This is because the arrow points the wrong direction. It will also fold a straight-flush, because that is included in *others*.

Better But Still Incomplete

```
When TripsOnBoard
  When NutFullHouseOrFourOfAKind < = 2 RaisePot force
  When HaveStraightFlush RaisePot force
  When others fold
```

Here you have fixed everything **except** you have still not accounted for the 0 value. Therefore the bot will still raise with any hand that is not a full house or quads as well as the top hands!

Right Way

```
When TripsOnBoard
  When NutFullHouseOrFourOfAKind < = 2 and (not NutFullHouseOrFourOfAKind = 0) RaisePot force
  When HaveStraightFlush RaisePot force
  When others fold
```

Position – the values for position are: first, middle, and last. You can only use an = sign to assign the value. This numerical-valued variable can be used preflop as well. We find that it covers play from every position very well.

When you have 4+ opponents and are in middle position, assigning *position = middle* only identifies you as "somewhere" in the middle. But in these post-flop situations your play is usually pretty cut and dried.

SuitsOnBoard – this is the number of different suits on the board, **not** the number of cards of the same suit on board. For example a flop of Js 9s 3h would be *SuitsOnBoard = 2* (or *SuitsOnBoard > 1)*.

Bet Size vs. Pot Size – we warned you earlier that this calculation is tricky and now here is the full explanation. Your opponent's uncalled bets are included in the pot size. This is necessary for being able to program simple actions based on pot size alone. However, it results in more complicated calculations for programming actions based on *BetSize* as a % of *PotSize*.

The easiest example to show you is when using **100% pot size** as your desired figure. For example, let's say you want to call in a certain situation if an opponent made less than a pot-sized bet. This one is simple because if the opponent bets 100% of the pot, then, when the action comes back to you, his bet **now** represents 50% of the entire pot -- so you could simply code:

```
When BetSize < 50% PotSize call force
```

Always keep in mind that if the amounttocall is < 50% of potsize it was less than a pot-sized bet, and consequently if amounttocall is over 50% of pot size the bet was more than the pot!

It is a bit more complicated, though, when calculating other pot size percentages. You cannot simply reduce them by half like you can with 100%. The actual formula for calculating this, if you care, is:

$$x = 1 / (1 + (1 / y))$$

Here is a handy chart for making the conversions on just about every bet size you are likely to want to use:

% of Pot Your Opponent Bet	Conversion (Use this for Bet Size % Potsize)
20%	17%
25%	20%
30%	23%
40%	29%
50%	33%
60%	38%
75%	43%
90%	47%
100%	50%
125%	56%
150%	60%
200%	67%

Get it? If an opponent bets 75% of the pot that bet is now included in the pot, so the amounttocall figure would be 43% of the current pot when it is your turn to act. This, of course, assumes no one else calls or rearises. A call by a third opponent before you can act reduces the amounttocall % of the potsize, so you should account for that by including a **calls =** figure in your coding in close decisions.

If an opponent over-bets the pot, say 125% of it (for instance if the pot was 18 and he bet 20) the amounttocall will be more than 50% of the current pot size when it's your turn to act. By the above chart you can see the amounttocall would be 56% of pot size in this case.

3.6.4 Previous Betting Round Action Variables

This group of variables deserves special attention. You can really get an edge over your fellow poker botters by taking prior betting round action into account in your decisions (of course our bots do this in their default programming as well). These are all Boolean variables which are either true or false. Here is the list:

RaisesBeforeFlop
BotRaisedBeforeFlop
BotIsLastRaiser

NoBettingOnFlop
FlushPossibleOnFlop
StraightPossibleOnFlop
PairOnFlop
CalledOnFlop (bot called on flop)
OpponentCalledOnFlop
TwoOfOneSuitPresentOnFlop
UncoordinatedFlop
MoreThanOneStraightPossibleOnFlop
AcePresentOnFlop
RaisesOnFlop
BotRaisedOnFlop

NoBettingOnTurn
FlushPossibleOnTurn
FourOf1SuiteOnTurn
StraightPossibleOnTurn
PairOnTurn
TripsOnBoardOnTurn
OneCardStraightPossibleOnTurn
MoreThanOneStraightPossibleOnTurn
RaisesOnTurn
OpponentCalledOnTurn
BotRaisedOnTurn

3.6.5 Defining Your Hand

Here is the list of Boolean variables for defining your hand strength.

HaveStraight
HaveNutStraight
Have2ndNutStraight
HaveUnderStraight
HaveStraightDraw
HaveNutStraightDraw
HaveInsideStraightDraw
HaveInsideNutStraightDraw
HavePair
HaveTopPair
Have2ndTopPair
Have3rdTopPair
Have4thTopPair
HaveBottomPair
HaveOverPair (any overpair)
Have2ndOverPair (pocket pair between top board card and 2nd top board card)
Have3rdOverPair (pocket pair between 2nd top board card and 3rd top board card)
Have4thOverPair (pocket pair between 3rd top board card and 4th top board card)
Have5thOverPair (pocket pair between 4th top board card and 5th top board card)
HaveUnderPair
HaveTwoPair
HaveTopTwoPair
HaveBottomTwoPair
HaveTrips
HaveSet
HaveTopSet
Have2ndTopSet
Have3rdTopSet
Have4thTopSet
HaveBottomSet
HaveQuads

HaveFullHouse
HaveStraightFlush
HaveNutStraightFlush
HaveFlush
HaveNutFlush
Have2ndNutFlush
Have3rdNutFlush
Have4thNutFlush
Have5thNutFlush
HaveFlushDraw
HaveNutFlushDraw
Have2ndNutFlushDraw
Have3rdNutFlushDraw
Have4thNutFlushDraw
Have5thNutFlushDraw
HaveBackdoorFlushDraw
HaveBackdoorNutFlushDraw
HaveBackdoor2ndNutFlushDraw
HaveBackdoor3rdNutFlushDraw
HaveNutFlushCard
HaveBestKickerOrBetter
PairInHand
HaveBestOverpairOrBetter
Have2ndBestOverpairOrBetter
Have3rdBestOverpairOrBetter
HaveBestKicker
Have2ndBestKicker
Have3rdBestKicker
Have2ndBestKickerOrBetter
Have3rdBestKickerOrBetter

Please Note: There cannot be any spaces in the above variables. This is one of two exceptions to the “spaces and line breaks don’t matter” rule (comments being the other). Please be careful, as it is natural to want to put a space after *have*. For example:

```
Have NutFlushDraw
```

..is incorrect and will return an error message from the bot. (This is an extremely common typo.)

Also: Certain hand-defining variables above include additional hand types. This is only logical. An obvious example would be *HavePair* which is also true when *HaveTwoPair*, *HaveSet*, or *HaveFullHouse* are true. Another example would be *HaveStraight* or *HaveFlush* which both are true when *HaveStraightFlush* is true.

Please review Appendix 1 carefully so that you understand the nuances of all the variables. For example *HaveTwoPair* is not true when one of those two pairs is on board. In that case you want to use *HaveOverPair* or maybe even *hand = AA* or *hand = KK*.

Overpair Variables Explained: Only the *HaveOverPair* variable actually refers to a hand that most poker players typically refer to as an "overpair." This is a bit counter-intuitive so make sure you understand it. *HaveOverPair* is any pocket pair that is higher in rank than all of the board cards. The rest of the overpair variables (*have2ndoverpair*, etc.) refer to other pocket pairs of varying strengths (see explanations above).

3.6.6 Using Sequential Order Post-flop

Preflop you learned that suited hands need to be coded before unsuited hands, and that all manner of exceptions (such as in the big blind or acting after you have been raised behind you) need to be coded ahead of regular actions. There is a similar Post-flop rule:

Code made hands before draws!

Get that one through your head also. From the bot's perspective a made flush or a straight is still a flush draw or a straight draw as well. Therefore it is imperative that you place the blocks of code covering playing straights and flushes higher than the blocks of code for playing drawing hands to flushes and straights. Likewise, if you have a code block that includes actions for *HaveFlush* or *HaveStraight* and that code block also has lines for *HaveFlushDraw* or *HaveStraightDraw*, place the commands for made hands higher in the block than the drawing hands.

If you place the codes for the drawing hands higher than the codes for their made hand counterparts, then the bot will find them first and act upon them even when you already have the made hand. The way you fix that is by paying attention to sequential order.

In fact paying attention to sequential order as a general practice just makes coding a heck of a lot easier. If you do this right you can save a lot of typing. Take a look at this example for handling a pocket pair in Limit Holdem when there is one over card on board:

```
When PairInHand and Overcards = 1
  When raises > 1 fold force
  When opponents > 1 and raises >= 1 fold force
  When opponents > 3 fold force
  When opponents = 3 and board = A fold force
  When opponents = 3 and (not board = A) call force
  When opponents <= 2 and bets = 0 and raises = 0 bet 60% force
  When opponents = 2 call force
  When opponents = 1 and raises = 1 and SuitsOnBoard = 2 call force
  When raises = 1 fold force
  When opponents = 1 bet force
```

Notice that I took a lot of shortcuts in the latter half of that code block which using sequential order properly allowed me to. For example the last two lines – I don't want to fold every time there is one raise, and I don't want to bet every time there is only 1 opponent - but I took care of the exceptions higher in the code block. So I could make those lines simple commands and it will still accomplish what I want.

3.6.7 Coding for Pot Odds

In NL games you normally handle pot odds by specifying *amounttocall* vs. *potsize* as covered earlier, or perhaps by assuming that small bets will usually be getting good implied odds for drawing hands.

In Fixed Limit, however, it is important to handle pot odds correctly because it's a huge component of the game. You need to zero in on profitable drawing situations and make sure you don't give opponents who are drawing free chances at it. There are a couple of methods for doing this.

The first way is by using the big blind count (as a numerical value assigned to pot odds). When using this method you must remember that on the flop the bet size is equal to 1 BB but on the Turn and River the bet size is equal to 2 BB.

Example

```
flop
when HaveNutFlushDraw and HavePair and not (HaveSet or HaveTrips or HaveFlush or HaveStraight or HaveTopPair or
HaveOverPair) and potsize >= 3 call force

turn
when HaveNutFlushDraw and HavePair and not (HaveSet or HaveTrips or HaveFlush or HaveStraight or HaveTopPair or
HaveOverPair) and potsize >= 7 call force
```

Another way is to use the *BetSize* vs. *PotSize* comparison, as has already been discussed. Remember to convert so that the bet is included in the pot size (although in Fixed Limit if you are using the chart we provided you can do it in reverse, as you are really only worried about staying away from the tiny pots).

Example

```
When HaveNutFlushDraw and HavePair and not (HaveSet or HaveTrips or HaveFlush or HaveStraight or HaveTopPair or
HaveOverPair)
  when BetSize <= 30% PotSize call force
```

Still another method is to use the opponent and call count (perhaps in conjunction with a previous betting round action variable) to arrive at a logical conclusion about the kind of odds you are getting. For example if there are 3+ opponents on the flop you know at least several small bets have gone in to the pot preflop. Similarly with 2 opponents on the Turn if there was raising on the flop you know the pot is pretty big. These types of assumptions are very reliable so you can effectively code for pot odds in Limit this way.

Examples

```
Flop
When opponents >=3 and not (FlushPossible or HavePair or HaveTrips or HaveStraight or HaveFullHouse) and HaveFlushDraw or
HaveStraightDraw call force

Turn
```

When opponents >=2 and (RaisesOnFlop or RaisesBeforeFlop) and not (FlushPossible or HavePair or HaveTrips or HaveStraight or HaveFullHouse) and HaveFlushDraw or HaveStraightDraw call force

Finally, you can use the *AmountToCall* vs. *PotSize* comparison to make simple calling decisions based on pot odds. Any money you have already bet in the round is included in the potsize calculation, so if you are getting the correct odds for a draw the decision should fairly simple.

Examples

Turn
When haveflushdraw and not (flushpossible or paironboard or havepair or havestraight) and amounttocall <= 20% potsize call force

When haveflushdraw and not (flushpossible or paironboard or havepair or havestraight) and amounttocall > 20% potsize fold force

When havenutstraightdraw and suitsonboard <= 3 and not (flushpossible or straightpossible or paironboard or havepair) and amounttocall <= 11% potsize call force

When havenutstraightdraw and suitsonboard <= 3 and not (flushpossible or straightpossible or paironboard or havepair) and amounttocall > 11% potsize fold force

When havenutstraightdraw and suitsonboard = 4 and not (straightpossible or paironboard or havepair or havestraight) and amounttocall <= 14% potsize call force

When havenutstraightdraw and suitsonboard = 4 and not (straightpossible or paironboard or havepair or havestraight) and amounttocall > 14% potsize fold force

3.6.8 Other Tips

We wanted to stick a section in here with additional tips that we can add to as the situations arise. This manual could only have been created with help from all the volunteer beta testers in our support forum, so a heartfelt thanks goes out to all of them. Interesting situations get discussed there on a daily basis. As the better coding tips are discovered, we will update this user guide with them here.

OpponentIsAllIn Tip – you should only use this variable in conjunction with *opponents = 1* because with another live opponent in the game you don't know their stack size or what they are going to do. This variable really only exists so you can call more liberally against one live opponent who has just gone all-in (as you know that their range is often pretty wide).

My favorite use for it is in tournaments when I have a large stack and will call with any hand (on any street except the river). For example:

When opponents = 1 and OpponentIsAllIn and BetSize < 12% StackSize call force

Bot's Last Action Tips – worth reiterating here; both of the variables *BotsLastAction* and *BotIsLastRaiser* carry backward to the last betting round. They can both be used to set up a condition from **either** the last betting round or the current betting round for handling raises after you act. When using them on the current betting round for handling subsequent raises, always remember that the bot's action is not recorded and therefore is not part of the *raises* or *calls* count.

So:

```
When BotsLastAction = raise and raises = 1
```

…means that at least one other player raised besides your poker bot.

RaisesSinceLastPlay Tip – Use this for Fixed Limit profiles when you want to program the bot to call one more raise but not two. Note that this numerical-valued variable can be used either preflop or post-flop. In Limit you will often want to fold for two raises cold but be getting good enough odds to always call one more raise.

Example

```
Preflop

When hand = KQ or hand = 99
  When raises = 0 and opponents <= 5 raise force
  When raises = 1 call force
  When RaisesSinceLastPlay > 1 fold force
  When RaisesSinceLastPlay = 1 call force
  When raises >= 2 fold force

Flop

When opponents = 2 and HaveStraightDraw and not (PairOnBoard or HaveStraight or HaveFlush or HaveFlushDraw or HavePair or
HaveTrips)
  When bets = 0 bet force
  When bets = 1 and raises = 0 call force
  When RaisesSinceLastPlay > 1 fold force
  When RaisesSinceLastPlay = 1 call force
  When raises > = 1 fold force
```

Notice how using sequential order intelligently in that code is required to get the desired results.

Limit Bet Size vs. Blind Size Tip – In NL games you only have the small blind and big blind to deal with as far as bet sizes go, and the minimum bet is always the big blind size. However in Limit there is a small bet size and a large bet size as well. Remember that the big blind size is equal to the small bet size in Limit, not the big bet size.

Programming a Stop-loss or Stop-win – If for some reason you are unwilling to risk your entire stack in a game, or just want to quit if you are having a losing session, you can program a stop-loss based on your starting stack size.

Example

```
Preflop
when startingstacksize >= 300% stacksize sitout force
```

Similarly, you can also quit when you reach a certain profit goal with the same type of code.

Example

```
Preflop
when startingstacksize <= 30% stacksize sitout force
```

TripsonBoard Tip – On the river if there are trips on board you may want to call against a single opponent with the top non-paired board card but fold with a worse full house. For example is the board is 99974 you may want to call a medium sized bet with A7 but not A4. To easily code for this situation you can use the variable *HaveTopNonBoardPairedPair*.

Opponent Stack Size Tip – We currently have seven numeric-valued variables that help you identify your opponent's stack size(s). Keep in mind only three of these variables deals with live opponent's current stack size. The rest are based on a "snapshot" of the stacks at the table before any cards were dealt and do not take into account who is still in the hand!

MaxOpponentStackSize – The number of big blinds that the opponent **at your table** with the largest stack has in his stack **at the beginning of the hand** (before blinds are posted). If there is only one opponent at your table this value will be the same as *MinOpponentStackSize*. The value is always 0 if you are playing at a poker room or game which does not support this feature. Can be used preflop or post-flop. Can be used on the left of the % comparators.

MinOpponentStackSize – The number of big blinds that the opponent **at your table** with the smallest stack has in his stack **at the beginning of the hand** (before blinds are posted). If there is only one opponent at your table this value will be the same as *MaxOpponentStackSize*. The value is always 0 if you are playing at a poker room or game which does not support this feature. Can be used preflop or post-flop. Can be used on the left of the % comparators.

OpponentsWithHigherStack – the number of opponents **at the table** with a larger stack than the bot's **at the beginning of the hand** (before blinds are posted). Equal stack sizes are not considered higher. The value is always 0 if you are playing at a poker room or game which does not support this feature. Can be used preflop or post-flop..

OpponentsWithLowerStack – the number of opponents **at the table** with a smaller stack than the bot's **at the beginning of the hand** (before blinds are posted). Equal stack sizes are not considered lower. The value is always 0 if you are playing at a poker room or game which does not support this feature. Can be used preflop or post-flop.

MaxCurrentOpponentStackSize – This is the good one: The number of big blinds that the **live opponent** (who is still in the hand and not yet folded) with the **largest current stack** has in his stack at the moment when it is the bot's turn to act. Can be used on the left of the % comparators to compare to your own stack size and is useful for identifying short-stacked opponents or determining that all the live opponents have your stack covered. Can be used preflop or post-flop.

MaxStillToActStackSize – The number of big blinds that the opponent with the largest stack **who has not acted yet** had in their stack at the beginning of the hand (before blinds were posted). Only valid on the first orbit (that is, the first time the betting goes around) for the betting round, otherwise the value is 0. Therefore it only applies to opponents behind you on the first orbit and is really **only accurate preflop**. Can be used to steal blinds from short-stacked opponents in tournament situations. Can be used on the left of the % comparators.

MinStillToActStackSize – The number of big blinds that the opponent with the lowest stack **who has not acted yet** had in their stack at the beginning of the hand (before blinds were posted). Only valid on the first orbit (that is, the first time the betting goes around) for the betting round, otherwise the value is 0. Therefore it only applies to opponents behind you on the first orbit and is really **only accurate preflop**. Can be used to identify large stacks in the blinds in tournament situations and stop normal steal-raises. Can be used on the left of the % comparators.

3.6.9 Example of a Post-flop NL Holdem Code

This example was written for NL cash games in mind. I usually use it with medium-tight preflop settings from the option menu and no preflop PPL code whatsoever. It isn't very long and usually does well, which shows you that you don't need a tremendous amount of code to get good results with our software. I present it here as a learning tool.

```
custom

flop

When TripsOnBoard and not (NutFullHouseOrFourOfAKind = 2 or NutFullHouseOrFourOfAKind = 1 or HaveQuads) and Betsize > 6 fold force

When PairOnBoard and not (FlushPossible or StraightPossible or board = A or board = K or board = Q) and raises = 0 and HaveTrips RaisePot force

When not (FlushPossible or StraightPossible) and HaveOverPair and (hand = AA or hand = KK) RaisePot force

When BotIsLastRaiser
  When (HaveFullHouse or HaveStraightFlush or HaveNutFlush or HaveTrips or HaveNutStraight or HaveSet) call force
  When opponents =2 and bets = 0 and (not FlushPossible) and not (board = A and StraightPossible) and not (board = K and StraightPossible) BetPot force
  When opponents = 1 and bets = 0 and raises = 0 and (not FlushPossible) BetPot force

When position = first
  When (HaveFullHouse or HaveStraightFlush or HaveNutFlush) call force
  When (HaveTrips or HaveNutStraight or HaveSet or HaveTwoPair) and SuitsOnBoard = 3 and bets = 0 call force
  When HaveTopPair and SuitsOnBoard = 2 and bets = 0 and raises = 0 BetMin force


When HaveOverPair
  When (HaveNutFlushDraw or HaveInsideStraightDraw) and bets >= 1 and Betsize < 55% Potsize RaiseMax force

When HaveTopPair
  When not (HaveFlushDraw or PairOnBoard or FlushPossible or RaisesBeforeFlop or Betsize > 50% Potsize) and raises = 0 RaisePot force

turn

When TripsOnBoard and not (NutFullHouseOrFourOfAKind = 2 or NutFullHouseOrFourOfAKind = 1 or HaveQuads or HaveStraightFlush) and BetSize > 6 fold force

When TwoPairOnBoard and not (NutFullHouseOrFourOfAKind = 2 or NutFullHouseOrFourOfAKind = 1 or NutFullHouseOrFourOfAKind = 3 or HaveQuads or HaveStraightFlush) and Betsize > 6 fold force

When QuadsOnBoard and not HaveBestKicker fold force
When QuadsOnBoard and HaveBestKicker call force

When OneCardFlushPossible and Betsize > 15% StackSize and (not HaveFlush) fold force

When PairOnBoard and not (FlushPossible or StraightPossible or board = A or board = K or board = Q) and HaveTrips RaisePot force

When PairOnBoard and not (FlushPossible or StraightPossible) and (board = A or board = K or board = Q) and HaveTrips and
HaveBestKicker and raises >= 1 call force

When opponents >=2 and position = first and bets = 0
  When not (FlushPossible or PairOnBoard) and HaveNutStraight call force
  When (not PairOnBoard) and HaveNutFlush call force
  When not (QuadsOnBoard or TripsOnBoard or TwoPairOnBoard) and HaveFullHouse call force
  When not (FlushPossible or PairOnBoard or StraightPossible) and HaveSet call force

When opponents >=2 and position = first and bets >= 1
  When not (FlushPossible or PairOnBoard) and HaveNutStraight RaiseMax force
  When (not PairOnBoard) and HaveNutFlush and (not StraightFlushPossibleByOthers) RaiseMax force
  When not (QuadsOnBoard or TripsOnBoard or TwoPairOnBoard) and HaveFullHouse RaiseMax force
  When not (FlushPossible or PairOnBoard or StraightPossible) and HaveSet RaiseMax force

When opponents = 1 and position = first and bets = 0 and raises = 0 and NoBettingOnFlop
  When not (OneCardFlushPossible or OneCardStraightPossible or HaveBestKickerOrBetter) and (hand = 2 or hand = 3 or hand = 4 or
hand = 5 or hand = 6 or hand = 7) Bet 70% force

When opponents = 1 and position = first and bets = 1 and raises = 0 and BetSize <= 7 and NoBettingOnFlop
  When not (OneCardFlushPossible or OneCardStraightPossible or PairOnFlop) and (hand = 8 or hand = 9) RaisePot force
  When Betsize < 30% potsize and HaveBestKickerOrBetter call force

When opponents = 1 and position = last and bets = 1 and raises = 0 and Betsize < 50% potsize and NoBettingOnFlop
  When not (PairOnBoard or OneCardFlushPossible or OneCardStraightPossible) and HaveFlushDraw RaiseMin force
  When PairOnFlop and not (OneCardFlushPossible or OneCardStraightPossible or HaveBestKickerOrBetter) and (hand = 2 or hand =
3 or hand = 4 or hand = 5 or hand = 6 or hand = 7) RaiseMin force

When opponents = 1 and position = last and bets = 1 and raises = 0
  When not (OneCardFlushPossible or OneCardStraightPossible) and HaveTrips and Overcards = 1 RaiseMax force
  When not (HavePair or HaveTwoPair or HaveQuads or HaveSet or HaveTrips or HaveFlush or HaveStraight or HaveFullHouse) and
(HaveFlushDraw or HaveStraightDraw) and Betsize > 50% potsize and Overcards <=1 fold force

When opponents = 1 and not (PairOnBoard or HaveFlush or HaveStraight or HaveSet or HaveTwoPair)
  When (HaveUnderPair or HaveBottomPair or have3rdtoppair) and (bets = 1 or raises = 1) fold force

When StraightPossible and RaisesOnFlop and not (PairOnBoard or FlushPossible or OneCardStraightPossible)
  When not (HaveTopPair or HaveTwoPair or HaveSet or HaveStraight) fold force

When HaveFlushDraw and not (PairOnBoard or FlushPossible or OneCardStraightPossible)
  When HaveTopPair and HaveBestKicker and raises >= 1 call force

river

When TripsOnBoard and not (NutFullHouseOrFourOfAKind = 1 or NutFullHouseOrFourOfAKind = 2 or HaveQuads or
HaveStraightFlush) and Betsize > 6 fold force
```

```
When TwoPairOnBoard and not (NutFullHouseOrFourOfAKind = 1 or NutFullHouseOrFourOfAKind = 2 or
NutFullHouseOrFourOfAKind = 3 or HaveQuads or HaveStraightFlush) and Betsize > 6 fold force

When QuadsOnBoard and not HaveBestKicker fold force

When QuadsOnBoard and HaveBestKicker RaiseMin force

When PairOnBoard and not (FlushPossible or StraightPossible or board = A or board = K or board = Q) and HaveTrips and raises = 1
and Betsize > 15 call force

When PairOnBoard and not (FlushPossible or StraightPossible) and (board = A or board = K or board = Q) and HaveTrips and
HaveBestKicker and raises = 1 call force

When opponents = 1 and FlushPossible and not (PairOnBoard or OneCardFlushPossible or OneCardStraightPossible)
  When HaveTopPair and (not HaveBestKicker) and bets = 0 call force
  When HaveTopPair and bets = 1 and Betsize < 100% potsize and Betsize < 15 call force

When opponents = 1 and PairOnBoard and StraightPossible and not (FlushPossible or OneCardStraightPossible)
  When HaveNutStraight and bets = 1 Raise 150% force
```

3.7 De-bugging for Typos

It’s a safe bet that your newly-written code is going to have typos, especially if you are going to create entire custom profiles. Thankfully, the bot will help you debug them. This is done by giving you error messages when you try to load a custom-coded profile that has bad code. The bot window will tell you which line the typo is on. This is done in sequential order, from top to bottom, until you get the entire code cleaned up.

So the first time you use the ‘Read Profile’ function for a new custom profile, one of two things will happen. Either it will read the entire profile and find no errors and load it successfully, or it will stop at the first typo it finds and return an error message. Something like this:

```
Line 87: Unrecognized character: #

Line 87: syntax error, unexpected NUMBER, expecting $end
Custom programming rejected due to errors
```

These error messages are printed in the bot’s log window and in the log file. If you get an error message like this, which is likely the first time reading a new profile, you need to fix the referenced typo and try again. If there are no further typos the profile will load successfully. If there are further typos you must keep repeating this process until they all get fixed and the profile finally loads.

3.7.1 Finding the Typo so You Can Correct It

The error message gives you the line that the typo is on. The way to go to that line is by using the *Go To* function in Notepad. Open the profile text file, go to Edit and click on Go To. If *Go To* it is grey and not clickable you need to go to Format and click on Word Wrap once and it should become clickable.

A small window will appear that is titled Goto Line. Enter the line number given in error message from the bot window (line 87 in the example above). Your curser will be placed at the beginning of the specified line, which will be at the bottom of the notepad window if it is a line below the text currently showing.

The error message from the bot should give you a pretty good idea of what the typo is.
Tip: Google “Notepad 2” and download it (it’s free) and all the codelines will always be numbered on the left.

3.7.2 Common Typos

• using the letter o instead of the number 0
• putting a space between <= or >=
• missing a parenthesis on one end
• misspelling terms such as *raise*, *OpponentsAtTable*, etc.
• forgetting to put *hand* = in front of each and every hand

… and others which you will undoubtedly invent.

Appendix 1

This Appendix lists all the numeric valued and Boolean valued variables supported by our software, along with their meanings and any special notes associated with the variable.

Numeric Valued Variables:

The following variables are used in conditions of the form <Variable> <Comparator> <Number>. For example:

```
Raises < 2
```

The complete list of numeric valued variables (in alphabetical order) is as follows:

AmountToCall – this is always expressed by number of big blinds, regardless of game type (Limit, NL, etc). It is the amount that the bot needs to put into the pot in order to call. It is 0 if the bot can check.

Bets – number of bets by opponents in this betting round. The bot's own bets are not counted. This variable is reset to 0 when flop, turn or river cards are dealt so that it represents the number of bets for the particular betting round only. The value can be either 0 or 1 since not more than 1 bet can be made in a particular betting round.

BetSize - this is always expressed by the number of big blinds, regardless of game type (Limit, NL, etc).

BigBlindSize - the size of the big blind expressed by betting units starting from 1, meant for tournaments.

BotsActionsOnThisRound – the number of actions taken on the current betting round which involve putting chips in the pot (calling or raising). **Checks are not counted** as an action.

BotsActionsOnFlop – the number of actions taken on the flop betting round which involve putting chips in the pot (calling or raising). **Checks are not counted** as an action. Restrictions: Turn and River only.

BotsActionsPreflop – the number of actions taken on the preflop betting round which involve putting chips in the pot (calling or raising). **Checks are not counted** as an action. Restrictions: Post-flop only.

BotsLastAction - whatever action the bot took last, good for handling raises after you act. This variable has its own special values which are: *none*, *beep*, *raise*, *bet*, *call*, or *check*. The only comparator allowed is '='.

BotsLastPreflopAction - whatever the bot's last action on the preflop betting round was. This variable uses the special BotsLastAction values which are: *none*, *beep*, *raise*, *bet*, *call*, or *check*. The only comparator allowed is '='. Restrictions: Post-flop only.

Calls – number of calls by opponents on this betting round. The bots' own calls are not counted. This variable is reset to 0 when flop, turn or river cards are dealt so that it represents the number of calls for the particular betting round only.

CallsSinceLastRaise – number of calls by all opponents since the last raise by an opponent on the current betting round. If there has not yet been a raise on the current betting round the value is 0. (The bot's own raise cannot be counted since it could never be the last raise.)

Checks – number of checks by opponents in this betting round. The bots' own checks are not counted. This variable is reset to 0 when flop, turn or river cards are dealt so that it represents the number of checks for the particular betting round only.

FirstCallerPosition – The position of the players at the table is counted from a counter-clockwise position relative to the big blind position (not your position). Therefore the small blind = 1, the button = 2, the cutoff (next to the button) = 3, etc. This variable tells you the position of the first preflop caller on the first round of betting when it is your turn to first act. It goes to 0 after the action passes and is therefore useless when the action comes back after a raise behind you. It is most useful when there has only been one caller and you are in the blinds or are in late position and want to know which position the call came from. Restrictions: First action preflop only.

FirstRaiserPosition – Works the same as FirstCallerPosition above except for identifying the position of a raiser (if any). Especially useful when there has been only one raise and you are in the blinds or are in late position and want to know which position it came from. Restrictions: First action preflop only.

Folds – number of folds since the current betting round card(s) were dealt.

LastCallerPosition – The position of the players at the table is counted from a counter-clockwise position relative to the big blind position (not your position). Therefore the small blind = 1, the button = 2, the cutoff (next to the button) = 3, etc. This variable tells you the position of the last preflop caller on the first round of betting when it is your turn to first act. It goes to 0 after the action passes and is therefore useless when the action comes back after a raise behind you. It is most useful when there has only been one caller and you are in the blinds or are in late position and want to know which position the call came from. Restrictions: First action preflop only.

LastRaiserPosition – Works the same as LastCallerPosition above except for identifying the position of a raiser (if any). Especially useful when there has been only one raise and you are in the blinds or are in late position and want to know which position it came from. Restrictions: First action preflop only.

LowCardsInHand – The number of unique low cards in your hand. An ace is counted as a low card and duplicates are not counted. For example in Omaha if our hand cards are AA39 this will evaluate to 2. For us to be able to make a low in Omaha Hi-Lo *LowCardsInHand* must be >= 2.

LowCardsOnBoard – The number of unique low cards on board. An ace is counted as a low card and duplicates are not counted. For example in Omaha/8 if the board on the turn is AA39 this will evaluate to 2. For a low to be possible in Omaha Hi-Lo *LowCardsOnBoard* must be >= 3. Restrictions: Post-flop only.

MaxCurrentOpponentStackSize – The number of big blinds that the **live opponent** (who is still in the hand and not yet folded) with the **largest current stack** has in his stack at the moment when it is the bot's turn to act. Can be used on the left of the % comparators to compare to your own stack size and is useful for identifying short-stacked opponents or determining that all the live opponents have your stack covered. Can be used preflop or post-flop.

MaxOpponentStackSize – The number of big blinds that the opponent at your table with the largest stack has in his stack at the beginning of the hand (before blinds are posted). The value is always 0 if you are playing at a poker room or game which does not support this feature. Can be used preflop or post-flop.

MinOpponentStackSize – The number of big blinds that the opponent at your table with the smallest stack has in his stack at the beginning of the hand (before blinds are posted). The value is always 0 if you are playing at a poker room or game which does not support this feature. Can be used preflop or post-flop.

MaxStillToActStackSize – The number of big blinds that the opponent with the largest stack **who has not acted yet** had in their stack at the beginning of the hand (before blinds were posted). Only valid on the first orbit (that is, the first time the betting goes around) for the betting round, otherwise the value is 0. Therefore it only applies to opponents behind you on the first orbit and is really only accurate preflop. Can be used to steal blinds from short-stacked opponents in tournament situations. Can be used on the left of the % comparators.

MinStillToActStackSize – The number of big blinds that the opponent with the lowest stack **who has not acted yet** had in their stack at the beginning of the hand (before blinds were posted). Only valid on the first orbit (that is, the first time the betting goes around) for the betting round, otherwise the value is 0. Therefore it only applies to opponents behind you on the first orbit and is really only accurate preflop. Can be used to identify large stacks in the blinds in tournament situations and stop normal steal-raises. Restrictions: Only valid on the first orbit of a betting round.

NumberOfRaisesBeforeFlop – The number of preflop raises made by opponents. The bot's own raises are not counted. Restrictions: Post-flop only.

NumberOfRaisesOnFlop – The number of raises on the flop betting round made by opponents. The bot's own raises are not counted. Restrictions: Turn and River only.

NumberOfRaisesOnTurn – The number of raises on the Turn betting round made by opponents. The bot's own raises are not counted. Restrictions: River only.

NutFullHouseOrFourOfAKind – This is 0 if we don't have a full house or four of a kind. Otherwise it returns a number indicating whether our hand is the 1st nut, 2nd nut and so on. If we have the 1st nut full house or four of a kind this returns 1. If we have the 2nd nut full house or 4 of a kind it returns 2 and so on. It does not take straight flushes into account. So it could return 1 even if our hand can be beaten by a straight flush. For example if we are holding an AK and the board cards on the flop are AAK then this will return 1 since we have the first nut hand with the AAAKK full house. Note that since we are holding an A a four of a kind is not possible for anybody and hence AAAKK is the 1st nut hand. However if we are holding KK and the board is AAK, the above variable evaluates to 3. This is because in this case AAAA is the 1st nut hand, AAAKK is the 2nd nut hand and our hand AAKKK is the 3rd nut hand. Restrictions: Post-flop only.

Opponent = - opponent screen names who still have cards in front of them on the current hand. This variable accepts user-created custom values which can consist of any combination of upper and lower case letters, numbers, dashes, and underscores. Only the equal sign can follow the word *opponent*. Use it to identify certain opponents whom you know, or whom you have stats on, and want to adjust your play for. You can group together large lists of opponent names separated by the 'or' operator inside of brackets. You can also set user-defined variables from those lists. This variable can be defined and used on any betting round. Restrictions: WPN Rooms only as of Dec 2016.

Opponents – the number of all live opponents, or everyone who has not yet folded (or has gone all-in on a prior betting round). This includes calls and raises in front of you as well. All-in players on a prior betting round are not counted.

OpponentsAtTable - only the starting number of players currently sitting in, useful for late stages of tournaments.

OpponentsLeft - same exact variable as 'opponents' above but describes it a little better (use either).

OpponentsOnFlop – the number of live opponents still in the hand at the beginning of the flop, before there was any betting action on the flop. Restrictions: Turn and River only.

OpponentsWithHigherStack – the number of opponents at the table with a larger stack than the bot's at the beginning of the hand (before blinds are posted). Equal stack sizes are not considered higher. The value is always 0 if you are playing at a poker room or game which does not support this feature. Can be used preflop or post-flop.

OpponentsWithLowerStack – the number of opponents at the table with a smaller stack than the bot's at the beginning of the hand (before blinds are posted). Equal stack sizes are not considered lower. The value is always 0 if you are playing at a poker room or game which does not support this feature. Can be used preflop or post-flop.

Overcards – The number of **hole cards** that are Overcards to the board (not vice-versa). Restrictions: Post-flop only.

OvercardsOnBoard – The number of **Board cards** that are Overcards to the hole cards. Please note that this includes paired board cards so is not necessarily just the number of higher ranks. Restrictions: Post-flop only.

Position - The bots' position relative to the other players. This variable has its own special values which are: *first*, *middle*, or *last*. The only comparator allowed is '='. **Your position is always read in real time whenever it is your turn to act.** For example if you are in middle position on the flop and the first opponent bets, you call, the last opponent raises, the original bettor now folds and it is back to you, now *opponents = 1 and position = first* will be true for the action you are facing. This variable is intended for post-flop use or second orbit situations pre-flop (when there was a raise behind you and the action has come back). For first orbit situations on the pre-flop betting round, use **stilltoact =** instead.

PotSize - this is always expressed by number of big blinds, regardless of game type (Limit, NL, etc).

Raises – number of raises by opponents on this betting round. The bots' own raises are not counted. This variable is reset to 0 when the flop, turn or river cards are dealt so that it represents the number of raises for the particular betting round only.

RaisesSinceLastPlay – The number of raises since bots' last action. Reset to 0 every time the bot beeps, checks, calls, bets, or raises and also when pre-flop, flop, turn, or river cards are dealt. This does not include the bots' own raises.

Random – returns a random number between 1 and 100, inclusive. Used to easily code randomized actions. For example: *when random <= 50* would be true 50% of the time on average.

StackSize - the size of the bots' chip stack counted by number of big blinds. (If the stack size is ever unknown due to temporary obstructions or connection issues then any condition that uses this variable will evaluate to false.)

StartingStackSize - the size of the bots' chip stack (counted by number of current big blinds) the first time a particular bot window reads it's stack size in the current session. Clicking start and stop on our bots will not reset this variable even if you move to another game, so you should close the bot window and reopen when you change tables. This variable should only be used in cash game profiles as it does not account for the increasing big blind size in tournaments. Also note that when the bot first reads it's stack size it is usually after a blind has been posted so the number associated with this variable will likely always be one big blind short. This variable is primarily meant to be used in the % comparators to compare it to StackSize so you can create stop-loss points if you so desire.

StillToAct - only the live players behind you who have not acted yet; does not include calls and raises in front of you. Only valid on the first orbit (that is, the first time the betting goes around) for the betting round. Therefore if somebody raises behind you and the betting comes back to you, *stilltoact* will always = 0.

SuitsInHand – The number of unique suits in your hole cards. This variable was primarily designed for our Omaha bot profiles. For example if your hole cards are Ad 3d 8d Js then this evaluates to 2 as you have only two suits in your jhand (diamonds and spades).

SuitsOnBoard – The number of unique suits on board. For example if the board cards are Ac 8c 7c on the flop then this evaluates to 1 as there is only 1 suite (clubs) on board. Restrictions: Post-flop only.

SuitsOnFlop – The number of unique suits that were on board on the flop only. For example if the flop came Ac 8c 7c on the flop then this evaluates to 1 as there was only 1 suite (clubs) on board. Restrictions: Post-flop only.

TotalInvested - the total amount of chips invested so far in the hand, counted by number of big blinds.

Boolean Valued Variables:

These variables evaluate to either *true* or *false* and form a condition by themselves. For example

When flushpossible

The complete list of Boolean valued variables (in alphabetical order) is as follows:

AcePresentOnFlop - True if there is an ace present on board which was dealt on the flop. Restrictions: Post-flop only.

BotCalledBeforeFlop – True if the bot called before the flop. This can be used pre-flop also and will be true if the bot has already called.

BotIsLastRaiser – True if the bot has raised and there have been no raises after that.

BotRaisedBeforeFlop – True if the bot raised before the flop. This can be used pre-flop also and will be true if the bot has already raised.

BotRaisedOnFlop – True if the bot raised on the flop. Restrictions: Post-flop only.

BotRaisedOnTurn – True if the bot raised on the turn. Restrictions: Turn and River only.

CalledOnFlop – True if the bot called on the flop. **Not true if the bot only checked**, bet or raised on the flop. However this is true even if the bot bet or raised, was raised or re-raised and then called on the flop. Restrictions: Post-flop only.

CalledOnTurn – True if the bot called on the turn. **Not true if the bot only checked**, bet or raised on the turn. However this is true even if the bot bet or raised, was raised or re-raised and then called on the turn. Restrictions: River only.

DoubleSuited – This variable is **no longer working**. Instead, to state that a pocket pair is double-suited use *hand = K suited K* suited, *hand = Q suited Q suited*, etc., which we have verified is working as desired. Please note that *suitsinhand = 2* is also true when you have three of 1 suit and 2 of another. Restrictions: Omaha Bots only.

FlushOnBoard - True if all the board cards on the River are the same suit. Restrictions: River only.

FlushPossible - True if a flush can be made with the current board cards. Restrictions: Post-flop only.

FlushPossibleOnFlop – True if a flush is/was possible on the flop. Restrictions: Post-flop only.

FlushPossibleOnTurn – True if a flush is/was possible on the turn. Restrictions: Turn and River only.

FourCardStraightInHand - True if the bot has 4 cards in sequence in the hole. For example this variable is *true* if the hand is 5678. Restrictions: Can only be true in Omaha games.

FourCardsToWheelOnBoard - True if a wheel can be made by using only one hole card. A wheel is A2345. Restrictions: Post-flop only.

FourCardsWith1GapInHand - True if the bot has 4 cards in sequence with 1 gap in the hole. For example this variable is *true* if the hand is 5689. Restrictions: Can only be true in Omaha games.

FourOf1SuitOnTurn – True if only 1 suite is/was present on the board on the turn. Restrictions: Turn and River only.

FullHouseOnBoard - True if there is a made full house on the board. Restrictions: Post-flop only.

HadPairOnFlop - True if the bot had any pair on the Flop, including a pocket pair. Restrictions: Turn and River only.

HadPairOnTurn - True if the bot had any pair on the Turn, including a pocket pair. Restrictions: River only.

HadTopPairOnFlop - True if the bot had top pair on the flop. Restrictions: Turn and River only.

HadTopPairOnTurn - True if the bot had top pair on the Turn. Restrictions: River only.

HadTwoPairOnFlop - True if the bot had two pair on the flop. Restrictions: Turn and River only.

HadOverpairOnFlop - True if the bot had any overpair on the flop. An overpair is defined as a pocket pair higher than all the board cards. Restrictions: Turn and River only.

Have10OutStraightDraw - True if the bot has a straight draw which has at least 10 outs to a straight. Can also be possible when the bot has a made straight already. Restrictions: Post-flop only. Can only be true in Omaha games.

Have12OutStraightDraw - True if the bot has a straight draw which has at least 12 outs to a straight. Can also be possible when the bot has a made straight already. Restrictions: Post-flop only. Can only be true in Omaha games.

Have16OutStraightDraw - True if the bot has a straight draw which has at least 16 outs to a straight. Can also be possible when the bot has a made straight already. Restrictions: Post-flop only. Can only be true in Omaha games.

Have2ndBestKicker - True if the bot has the 2nd best kicker or the best kicker with the current board cards (to clarify, this variable is also true when *havebestkicker* is true). The 2nd best kicker is defined as the 2nd highest valued card not currently present on the board. The best kicker is defined as the highest valued card not currently present on the board. Ace is highest. Restrictions: Post-flop only.

Have2ndBestKickerOrBetter - True if the bot has a made hand which is 2nd best kicker or better given the current board cards. The “or better” part refers to any better hand values such as a pair, two pair, etc and not just a better kicker. Restrictions: Post-flop only.

Have2ndBestOverpairOrBetter - True if the bot has a made hand which is 2nd best overpair or better given the current board cards. 2nd best overpair is KK with no A or K on board. The “or better” part refers to any better hand values such as a pair, two pair, etc and not just a better overpair. Restrictions: Post-flop only.

Have2ndNutFlush - True if the bot has the 2nd highest possible flush with the current board cards. For Holdem the 2nd highest possible flush could consist of all cards from the board. Note that 2nd nut flushes could be beaten by a higher flush, full houses, four of a kind and straight flushes. Restrictions: Post-flop only.

Have2ndNutFlushDraw - True if the bot has a made 2nd nut flush or a 2nd nut flush draw. Restrictions: Post-flop only.

Have2ndNutLow - True if the bot has the 2nd best possible low using the current board cards. Restrictions: Post-flop only. Can only be true in Omaha/8 games.

Have2ndNutLowDraw - True if the bot has the 2nd nut low draw. If a low is possible then this variable will be *false*. This variable is also *false* for made lows. Restrictions: Post-flop only. Can only be true in Omaha/8 games.

Have2ndNutStraight - True if the bot has the 2nd best possible straight using the current board cards. Note that this may not be the 2nd best hand if a flush or full house is possible. Restrictions: Post-flop only.

Have2ndOverPair - True if the bot has a hole pair which is between the highest board card and the 2nd highest card rank on board and therefore is not technically an overpair to the board. Ace is highest. Pairs on board are only counted as one rank. Restrictions: Post-flop only.

Have2ndTopPair - True if the bot has a hole card that is the same value as the 2nd highest card on board. Ace is highest. Restrictions: Post-flop only.

Have2ndTopSet - True if the bot has a hole pair that has the same value as the 2nd highest board card. Ace is highest. Restrictions: Post-flop only.

Have3rdBestKicker - True if the bot has the 3rd best kicker, the 2nd best kicker, or the best kicker with the current board cards (to clarify, this variable is also true when *havebestkicker* or when *have2ndbestkicker* is true). The 3rd best kicker is defined as the 3rd highest valued card not currently present on the board. The 2nd best kicker is defined as the 2nd highest valued card not currently present on the board. The best kicker is defined as the highest valued card not currently present on the board. Ace is highest. Restrictions: Post-flop only.

Have3rdBestKickerOrBetter - True if the bot has a made hand which is 3rd best kicker or better given the current board cards. The “or better” part refers to any better hand values such as a pair, two pair, etc and not just a better kicker. Restrictions: Post-flop only.

Have3rdBestOverpairOrBetter - True if the bot has a made hand which is 3rd best overpair or better given the current board cards. 3rd best overpair is QQ with no A, K or Q on board. The “or better” part refers to any better hand values such as a pair, two pair, etc and not just a better overpair. Restrictions: Post-flop only.

Have3rdNutFlush - True if the bot has the 3rd highest possible flush with the current board cards. For Holdem the 3rd highest possible flush could consist of all cards from the board. Note that 3rd nut flushes could be beaten by a higher flush, full houses, four of a kind and straight flushes. Restrictions: Post-flop only.

Have3rdNutFlushDraw - True if the bot has a made 3rd nut flush or a 3rd nut flush draw. Restrictions: Post-flop only.

Have3rdNutLow - True if the bot has the 3rd best possible low using the current board cards. Restrictions: Post-flop only. Can only be true in Omaha/8 games.

Have3rdNutLowDraw - True if the bot has the 3rd nut low draw. If a low is possible then this variable will be *false*. This variable is also *false* for made lows. Restrictions: Post-flop only. Can only be true in Omaha/8 games.

Have3rdOverPair - True if the bot has a hole pair which is between the 2nd highest and 3rd highest card rank on board and therefore is not technically an overpair to the board. Ace is highest. Pairs on board are only counted as one rank. Restrictions: Post-flop only.

Have3rdTopPair - True if the bot has a hole card that is the same value as the 3rd highest card on board. Ace is highest. Restrictions: Post-flop only.

Have3rdTopSet - True if the bot has a hole pair that has the same value as the 3rd highest board card. Ace is highest. Restrictions: Post-flop only.

Have4thNutFlush - True if the bot has the 4th highest possible flush with the current board cards. For Holdem the 4th highest possible flush could consist of all cards from the board. Note that 4th nut flushes could be beaten by a higher flush, full houses, four of a kind and straight flushes. Restrictions: Post-flop only.

Have4thNutFlushDraw - True if the bot has a made 4th nut flush or a 4th nut flush draw. Restrictions: Post-flop only.

Have4thNutLow - True if the bot has the 4th best possible low using the current board cards. Restrictions: Post-flop only.

Have4thNutLowDraw - True if the bot has the 4th nut low draw. If a low is possible then this variable will be *false*. This variable is also *false* for made lows. Restrictions: Post-flop only. Can only be true in Omaha/8 games.

Have4thOverPair - True if the bot has a hole pair which is between the 3rd highest and 4th highest card rank on board and therefore is not technically an overpair to the board. Ace is highest. Pairs on board are only counted as one rank. Restrictions: Post-flop only.

Have4thTopPair - True if the bot has a hole card that is the same value as the 4th highest card on board. Ace is highest. Restrictions: Post-flop only.

Have4thTopSet - True if the bot has a hole pair that has the same value as the 4th highest board card. Ace is highest. Restrictions: Turn and River only.

Have5thNutFlush - True if the bot has the 5th highest possible flush with the current board cards. For Holdem the 5th highest possible flush could consist of all cards from the board. Note that 5th nut flushes could be beaten by a higher flush, full houses, four of a kind and straight flushes. Restrictions: Post-flop only.

Have5thNutFlushDraw - True if the bot has a made 5th nut flush or a 5th nut flush draw. Restrictions: Post-flop only.

Have5thOverPair - True if the bot has a hole pair which is between the 4th highest and 5th highest card rank on board, so is not technically an overpair to the board. Ace is highest. Pairs on board are only counted as one rank. Restrictions: Post-flop only.

HaveBackdoor2ndNutFlushDraw - True if the bot has a made 2nd nut flush or a 2nd nut flush draw or a backdoor 2nd nut flush draw. A backdoor flush draw is defined as a hand that has three playable cards of a particular suit so that getting two more cards of the same suit can complete the flush. Restrictions: Flop only.

HaveBackdoor3rdNutFlushDraw - True if the bot has a made 3rd nut flush or a 3rd nut flush draw or a backdoor 3rd nut flush draw. A backdoor flush draw is defined as a hand that has three cards of a particular suit including the so that getting two more cards of the same suite can complete the flush. Restrictions: Flop only.

HaveBackdoorFlushDraw - True if the bot has a made flush or a flush draw or a backdoor flush draw. A backdoor flush draw is defined as a hand that has three cards of a particular suit so that getting two more cards of the same suit can complete the flush. Restrictions: Flop only.

HaveBackdoorNutFlushDraw - True if the bot has a made nut flush or a nut flush draw or a backdoor nut flush draw. A backdoor flush draw is defined as a hand that has three cards of a particular suit so that getting two more cards of the same suite can complete the flush. Restrictions: Flop only.

HaveBestKicker - True if the bot has the best kicker with the current board cards. Best kicker is defined as the highest valued card not currently present on the board. Ace is highest. Restrictions: Post-flop only.

HaveBestKickerOrBetter - True if the bot has a made hand which is best kicker or better given the current board cards. Note that the "or better" part of this variable refers to any higher hand rank, including a pair, set, straight, full house, etc. Restrictions: Post-flop only.

HaveBestOverpairOrBetter - True if the bot has a made hand which is best overpair or better given the current board cards. Best overpair is AA with no A on board. Note that the "or better" part of this variable refers to any higher hand rank, including a pair, set, straight, full house, etc. Restrictions: Post-flop only.

HaveBottomPair - True if the bot has a hole card that is the same value as the lowest card on board. Ace is highest. Restrictions: Post-flop only.

HaveBottomSet - True if the bot has a hole pair that has the same value as the lowest board card. Ace is highest. Restrictions: Post-flop only.

HaveBottomTwoPair - True if the two lowest valued board cards are present in the hole for the bot. A pair on board does not count as one of the pairs. Ace is highest. Restrictions: Post-flop only.

HaveFlush - True if the bot has a made flush. For Holdem a flush on board **does** count as a made flush. Restrictions: Post-flop only.

HaveFlushDraw - True if the bot has a made flush or a flush draw. Restrictions: Post-flop only.

HaveFullHouse - True if the bot has a made full house. Any playable full house, can consist of only one card in the case of two pair on board or play the board on the river if a full house in on board. Restrictions: Post-flop only.

HaveInsideNutStraightDraw - True if the bot has an inside nut straight draw. An inside nut straight draw is defined as a hand with at least 4 'outs' to a nut straight. Unlike *NutStraightDraw,* outs that make a flush possible are **not** excluded. Note that this condition is also true when *HaveNutStraightDraw* is true. Restrictions: Post-flop only.

HaveInsideStraightDraw - True if the bot has a made straight or an inside straight draw. An inside straight draw is defined as a hand with at least 4 'outs' to a straight. Note that this condition is also true when *HaveStraightDraw* is true Restrictions: Post-flop only.

HaveLow - True if the bot has a made low using the current board cards. Restrictions: Post-flop only. Can only be true in Omaha/8 games.

HaveNothing - True if the bot has no pair, no trips, no straight, no straight draw, no inside straight draw, no flush, no flush draw, no backdoor flush draw; and if overcards = 2 is also false. This variable only applies to high hands in Omaha Hi-Lo and disregards made low hands or low draws, so should **not** be used in Omaha Hi-Lo. Restrictions: Post-flop only.

HaveNutFlush - True if the bot has the highest possible flush with the current board cards. For Holdem a royal straight flush on board **does** count as a made nut flush. Note that nut flushes could be beaten by full houses, four of a kind and straight flushes. Restrictions: Post-flop only.

HaveNutFlushCard - True if a flush is possible based on the current board cards and one of the bot's hole cards is the card required for a nut flush. The bot may not have a made flush but since it is holding the nut flush card nobody else can have the nut flush either. For example if the board is Ac Qc 3c then the nut flush card is Kc. Restrictions: Post-flop only.

HaveNutFlushDraw - True if the bot has a made nut flush or a nut flush draw. Restrictions: Post-flop only.

HaveNutLow - True if the bot has the best possible low using the current board cards. Restrictions: Post-flop only. Can only be true in Omaha/8 games.

HaveNutLowBackdoorDraw - True if the bot has the nut low backdoor draw. This variable is true only if there is only 1 low card on board such that if two new low cards are dealt out (without counterfeiting the bots' low cards) then the bot will have the nut low on the river. Restrictions: Flop only. Can only be true in Omaha/8 games.

HaveNutLowDraw - True if the bot has the nut low draw. If a low is possible then this variable will be *false*. This variable is also *false* for made lows. Restrictions: Post-flop only. Can only be true in Omaha/8 games.

HaveNutLowDrawWithBackup - True if the bot has the nut low draw with backup. This variable will be *false* if a low is already possible. A backup implies that any non duplicate low card being dealt will cause the bot to have the nut low. For example if the board is 67T and the bots hole cards are A29T then the bot has a nut low draw but no backup since if an A or 2 is dealt on the turn the bot will not have the nut low. Therefore this variable will be *false* in this situation. However if the bots' hold cards in this situation are A23T then the bot will have the nut low on the turn if any non duplicate low card is dealt. So even if an A, 2 or a 3 is dealt on the turn the bot will have the nut low. Therefore this variable will be *true* in this situation. Restrictions: Post-flop only. Can only be true in Omaha/8 games.

HaveNutLowWithBackup - True if the bot has the best possible low using the current board cards with backup. A backup implies that the nut low cannot be counterfeited by a new board card. For example if the board is 678 and the bots' hole cards are A29T then the bot has the nut low but no backup because if an A or a 2 is dealt on the turn the bot will no longer have the nut low. Therefore this variable will be *false* in this situation. However if the bots' hole cards are A23T then the bot has the nut low with backup. Even if an A or a 2 is dealt on the turn the bot will still have the nut low. Therefore this variable will be *true* in this situation. Restrictions: Post-flop only. Can only be true in Omaha/8 games.

HaveNuts - True if the bot has the best possible hand at the moment. This variable does **not** apply to low hands in Omaha Hi-Lo, only the best possible high hand. Restrictions: Post-flop only.

HaveNutStraight - True if the bot has the best possible straight using the current board cards. Note that this may not be the nut hand if a flush or a full house is possible. Restrictions: Post-flop only.

HaveNutStraightDraw - True if the bot has a made nut straight or has a nut straight draw. A nut straight draw is defined as a hand with at least 7 'outs' to a nut straight. If a flush is not already possible then a card that will complete a nut straight but also make a flush possible is **not** counted as an 'out' to a nut straight. Restrictions: Post-flop only.

HaveNutStraightFlush - True if the bot has a made straight flush and it is not possible for an opponent to have a higher straight flush. For Holdem a royal straight flush on board **does** count as a made nut straight flush. Restrictions: Post-flop only.

HaveOverPair - True if the bot has a hole pair that is higher in value than any card on board. Ace is highest. Restrictions: Post-flop only.

HavePair - True if the bot has a pair. This could be a pair in hand or one of the hand cards paired with at least one of the cards on board. A pair on board does **not** count. Restrictions: Post-flop only.

HaveQuads - True if the bot has a made four of a kind. Four of a kind on board does **not** count. Restrictions: Post-flop only.

HaveSet - True if the bot has a hole pair that has the same value as one of the board cards. Restrictions: Post-flop only.

HaveStraight - True if the bot has any made straight. Restrictions: Post-flop only.

HaveStraightDraw - True if the bot has a made straight or a straight draw. A straight draw is defined as a hand with at least 7 'outs' to a straight. Cards that could complete a flush **are** counted as outs. Restrictions: Post-flop only.

HaveStraightFlush - True if the bot has a made straight flush. For Holdem a straight flush on board **does** count as a made straight flush. Restrictions: Post-flop only.

HaveTopNonBoardPairedPair - True if the bot has a hole card that is the same value as the highest non-paired card on board. Ace is highest. This was designed to identify the strength of two-pair hands when a pair is on board. (Please note that *HaveTwoPair* in PPL cannot include a board pair however). Restrictions: Post-flop only.

HaveTopPair - True if the bot has a hole card that is the same value as the highest card on board. Ace is highest. This could include a paired board card and would therefore be true if you have top trips as well. Restrictions: Post-flop only.

HaveTopSet - True if the bot has a hole pair that has the same value as the highest board card. Ace is highest. Restrictions: Post-flop only.

HaveTopTwoPair - True if the two highest valued board cards are present in the hole for the bot. A pair on board does **not** count as one of the pairs. Ace is highest. Restrictions: Post-flop only.

HaveTrips - True if there is a pair on board and the bot has a hole card of the same value as the board pair. Three of a kind on board does **not** count as having trips. Restrictions: Post-flop only.

HaveTwoPair - True if the bot has two pair. A pair on board does **not** count as one of the pairs. Restrictions: Post-flop only.

HaveUnderPair - True if the bot has a hole pair (and the highest hole pair) is lower in value than the lowest card on board. Ace is highest. Restrictions: Post-flop only.

HaveUnderStraight - True if the bot has a straight such that all hand cards used to make the straight are lower in value than any board cards used to make the straight. Restrictions: Post-flop only.

IsFinalTable - True if the bot is playing at the Final Table at an MTT, which has a different appearance than the other tables. Our bot software recognizes this and has no problem playing at the final table; however you may want to program different instructions for it by using this variable. Restrictions: Not available at any currently supported poker rooms as of 10-15-2010.

KingPresentOnFlop - True if there is a king present on board which was dealt on the flop. Restrictions: Post-flop only.

LowPossible - True if a low can be made with the current board cards. Restrictions: Post-flop only. Can only be true in Omaha/8 games.

LowPossibleOnFlop – True if a low is/was possible on the flop. Restrictions: Post-flop only. Can only be true in Omaha/8 games.

LowPossibleOnTurn – True if a low is/was possible on the turn. Restrictions: Omaha only, Turn and River only. Can only be true in Omaha/8 games.

MoreThanOneStraightPossibleOnFlop - True if there is/was more than one way to make a straight on the flop. Restrictions: Post-flop only.

MoreThanOneStraightPossibleOnTurn – True if there is/was more than one way to make a straight on the turn. Restrictions: Turn and River only.

NoBettingOnFlop – True if there was no betting on the flop. I.e. everybody checked on the flop. However note that as a **special case only for Omaha Hi-Lo** this is also true if the bot was in the last position on the flop, it was checked around to the bot, the bot bet and after that there were no raises. Restrictions: Turn and River only.

NoBettingOnTurn – True if there was no betting on the turn. I.e. everybody checked on the turn. However note that as a **special case only for Omaha Hi-Lo** this is also true if the bot was in the last position on the turn, it was checked around to the bot, the bot bet and after that there were no raises. Restrictions: River only.

OneCardFlushPossible - True if a flush can be made by using one hole card with the current board cards. Restrictions: Post-flop only.

OneCardStraightPossible - True if a straight can be made by using one hole card with the current board cards. Restrictions: Post-flop only.

OneCardStraightFlushPossible - True if a straight flush can be made by using one hole card with the current board cards. Restrictions: Post-flop only.

OneCardStraightPossibleOnTurn – True if a one card straight is/was possible on the turn. Restrictions: Turn and River only.

Only1OneCardStraightPossible - True if one and only one straight can be made using only one hole card with the current board cards. For example if the board is 4578 then the only one-card straight possible is with a 6 so this variable will be *true* in this situation. However if the board is 4567 then there are two one-card straights possible and this variable will be *false* in this situation. Restrictions: Post-flop only.

OnlyOneStraightPossible - True if one and only one straight can be made with the current board cards. For example if the board is AKQ then the only possible way to make a straight is with JT and so this variable will be *true* in this situation. However if the board is KQJ then the two ways to make a straight are with an AT or with a T9 so this variable will be *false* in this situation. Restrictions: Post-flop only.

OpponentCalledOnFlop – True if at least one opponent called on the flop **and** no opponent bet or raised. Restrictions: Turn and River only.

OpponentCalledOnTurn – True if at least one opponent called on the turn **and** no opponent bet or raised. Restrictions: River only.

OpponentIsAllin – True if any opponent has gone all-in at any point in the current hand, including the current betting round.

Others – Always True

PairInHand - True if the bot has a hole pair. Also true if the bot has two pair, trips, or quads in the hole.

PairOnBoard - True if there is a pair on board. This will also be true if there are two pair, trips, quads, or a full house on board. Restrictions: Post-flop only.

PairOnFlop – True if there is/was a pair on board on the flop. Restrictions: Post-flop only.

PairOnTurn – True if there is/was a pair on board on the turn. Restrictions: Turn and River only.

QuadsOnBoard - True if there is a four of a kind on board. Restrictions: Post-flop only.
QueenPresentOnFlop - True if there is a queen present on board which was dealt on the flop. Restrictions: Post-flop only.

RaisesBeforeFlop – This is true if any opponent raised before the flop. Raises by the bot are not counted. If you want to check whether the bot raised before the flop use the variable *BotRaisedBeforeFlop* instead. Restrictions: Post-flop only.

RaisesOnFlop - True if any opponent raised on the flop. The bots' raises are not counted. Restrictions: Turn and River only.

RaisesOnTurn - True if any opponent raised on the turn. The bots' raises are not counted. Restrictions: River only.

RiverCardisOvercardToBoard – True if the card that was dealt on the River was higher in rank than all the other board cards. Restrictions: River only.

SecondTopFlopCardPairedonRiver - True if the second-highest ranking board card on the flop paired on the River. Restrictions: River only.

SecondTopFlopCardPairedonTurn - True if the second-highest ranking board card on the flop paired on the Turn. Restrictions: Turn and River only.

StackUnknown - True if the bot cannot read its own stack size and therefore returns a "stack unknown" message in the bot window. Potentially useful for making sure the bot does not get involved with anything but strong hands when it is temporarily not reading everything properly.

StraightFlushPossible - True if a straight-flush can be made with the current board cards. Restrictions: Post-flop only.

StraightFlushPossibleByOthers - True if a straight-flush can be made by an opponent with the current board cards given the bots hole cards. The bot may or may not have a straight flush and a straight flush may be possible with the current board cards but the bots' hole cards may rule out a straight flush for anybody else. For example if the board is Ac Kc Qc and the bot is holding Tc and 4c then the bot does not have a straight flush but nobody else can have a straight flush either because the bot holds Tc which is required to make a straight flush with the current board cards. This variable will evaluate to *false* while *StraightFlushPossible* will evaluate to *true*. Therefore the bots' nut flush hand cannot be beaten by a straight flush in this situation. Restrictions: Post-flop only.

StraightOnBoard - True if there is a straight on board. Restrictions: Post-flop only.

StraightPossible - True if a straight can be made with the current board cards. Restrictions: Post-flop only.

StraightPossibleOnFlop – True if straight is/was possible on the flop. Restrictions: Post-flop only.

StraightPossibleOnTurn – True if straight is/was possible on the turn. Restrictions: Turn and River only.

ThreeCardStraightInHand - True if the bot has three consecutive ranks in it's hole cards. Restrictions: Can only be true in Omaha games.

ThreeCardStraightOnBoard - True if there are three consecutive ranks on the board. Restrictions: Post-flop only.

ThreeCardsWith1gapInHand - True if the bot has three cards in sequence with 1 gap in the hole. For example this variable is *true* if the hand is 568A. Restrictions: Can only be true in Omaha games.

TopFlopCardPairedonRiver - True if the highest ranking board card on the flop paired on the River. Restrictions: River only.

TopFlopCardPairedonTurn - True if the highest ranking board card on the flop paired on the Turn. Restrictions: Turn and River only.

TripsInHand - True if the bot has trips in the hole. Restrictions: Can only be true in Omaha games.

TripsOnBoard - True if there is a three of a kind on board. This will also be true if there is a full house or quads on board. Restrictions: Post-flop only.

TripsOnBoardOnTurn – True if three of a kind is/was on board on the Turn. This will also be true if a full house or quads is/was on board on the Turn. Restrictions: Turn and River only.

TurnCardPaired – True if the card that was dealt on the Turn was paired on the River. It must be the card that was dealt on the Turn – not true if any other card pairs on the River. Restrictions: River only.

TurnCardisOvercardToBoard – True if the card that was dealt on the Turn was higher in rank than all the other board cards. Restrictions: Turn and River only.

TwoPairInHand - True if the bot has two pair in the hole. Restrictions: Can only be true in Omaha games.

TwoPairOnBoard - True if there are two pair on board. This will also be true if there is a full house on board. Restrictions: Post-flop only.

UncoordinatedFlop – True if the flop contains/contained no pair on board, no possible flush, three different suits, no possible straight, and the ranks are such that no opponent could have 7 or more outs to a straight. Restrictions: Post-flop only.

User[a-z_0-9] – Set when a situation that is defined by the user is read in the code, with the action being that of setting a user-defined variable instead of taking a normal action (see section 3.2.5). In this case the bot keeps reading code until a matching condition with a normal executable action is found (bet, raise, check, fold, beep, sitout, etc.). True when the condition that the user-variable defined earlier in the code is found true. Only letters, numbers, and underscores can be used in the variable name; however letters are not case-sensitive. **Notes**: All user defined variables are set to false when cards are dealt for a new hand. Once set to true they retain their value until cards are dealt for a new hand. Therefore variables set to true during early betting rounds can be used on later streets. Situations can be defined to set a user-variable the same way any other situation is defined, using any of the existing PPL variables (both numeric and Boolean). Once the user-variable is set it is used like any other Boolean variable. See section 3.2.5 for examples.

WheelPossible - True if a wheel can be made with the current board cards. A wheel is A2345. Restrictions: Post-flop only.

Appendix 2: Example of an Omaha/8 profile

This Appendix gives a working example of a PPL-coded profile written for Pot-Limit Omaha Hi-Lo tournaments. It is meant for instructional purposes but can be used as-is and in fact has done very well in large-field MTT's. It just needs to be added to a saved profile underneath where your option setting choices display and loaded after starting the bot using the Read Profile menu item.

```
custom
Preflop

When opponents = 1 and OpponentIsAllIn and BetSize < 20% StackSize call force

When StackSize > 100 and raises >= 1 and Betsize > 12 and not (hand = AA or hand = Asuited2 or hand = Asuited34 or hand =
Asuited35 or hand = Asuited36 or hand = Asuited3KQ or hand = Asuited3KJ or hand = Asuited3QJ or hand = Asuited3KT or hand =
Asuited3QT or hand = A suited 3 J T or hand = A suited KK or hand = A23 or hand = A24) fold force

When raises >= 1 and BetSize > 4 and hand = AA RaisePot force

When OpponentsAtTable = 1  and (hand = A  or hand = KK  or hand = QQ or hand = JJ or hand = KQJT  or hand = KQJ9  or hand =
KQT9  or hand = KJT9  or hand = KQJT or hand = KQJ9  or hand = KQT9 or hand = 234  or hand = 235  or hand = 245  or hand =
246 ) and not (hand = KKK or hand = QQQ or hand = JJJ) RaisePot force

When OpponentsAtTable = 2 and BotsLastAction = raise and (hand = AA or hand = A 2 or hand = A suited) RaisePot force
When OpponentsAtTable = 2  and ( hand = A suited  or hand = KK  or hand = QQ  or hand = A 4  or hand = A 5  or hand = 2 3 4  or
hand = 2 3 5  or hand = 2 3 K suited  or hand = 2 4 5  or hand = 2 3 6  or hand = 2 4 5 6 ) and not (hand = KKK or hand = QQQ)
RaiseMin force

When In BigBlind and BetSize < 1 and StackSize > 15 call force

When In SmallBlind and StackSize > 15 and raises = 0 and calls >= 1 and (hand = A  or hand = KK  or hand = QQ or hand = JJ or
hand = KQJT  or hand = KQJ9  or hand = KQT9  or hand = KJT9  or hand = KQJT or hand = KQJ9  or hand = KQT9 or hand = 234
or hand = 235  or hand = 245  or hand = 246 or TwoPairInHand) call force

When TripsInHand and not (hand = AA) fold force

When StackSize > 75 and raises >= 1 and BetSize > 10
  When not (hand = AA or hand = A2 or hand = Asuited34 or hand = Asuited35 or hand = Asuited36 or hand = Asuited45K or hand =
A34K or hand = A35Ksuited or hand = A36Ksuited or hand = AKsuited45 or hand = A345 or hand = A346 or hand = Asuited456 or
hand = A34Qsuited or hand = A35Qsuited or hand = KKQQ or hand = A KK suited) fold force
  When not (hand = A suited or hand = J suited) and (hand = A289 or hand = A228 or hand = A229 or hand = A22T or hand = A28T
or hand = A29T or hand = A222 or hand = A279 or hand = A27T or hand = A27J or hand = A28J or hand = A29J) fold force

When StackSize <= 12
  When ( hand = AA  or hand = A 2  or hand = A suited 3  or hand = A 3 4  or hand = A 3 5  or hand = A 3 6  or hand = A 3 K suited
or hand = A 4 5  or hand = A suited 4 6  or hand = A suited 5 6 7  or hand = A K suited 4 6  or hand = KK QQ  or hand = KK JJ  or
hand = QQ JJ  or hand = A suited K Q J  or hand = A suited K Q T  or hand = A suited K J T  or hand = A suited Q J T  or hand = A
KK  or hand = A suited QQ) and not (hand = KKK or hand = QQQ) RaisePot force
```

```
When StackSize <= 8
  When ( hand = A  or hand = KK  or hand = QQ  or hand = 2 3 4  or hand = 2 3 5  or hand = 2 4 5  or hand = 2 4 6  or hand = K suited Q J T  or hand = K suited Q J 9  or hand = K suited Q T 9  or hand = K suited J T 9  or hand = K Q suited J T  or hand = K Q suited J 9  or hand = K Q suited T 9 or TwoPairInHand) and not (hand = KKK or hand = QQQ) RaisePot force

When StackSize <= 6
  When BotsLastAction = raise RaisePot force

When Betsize > 25% StackSize
  When hand = AA RaisePot force

When BetSize > 12
  When (hand = AA 2 or hand = AA 3 or hand = AA 4 or hand = AA suited) RaisePot force

When BetSize > 20% StackSize and StackSize > 50
  When not ( hand = AA 2  or hand = AA 3  or hand = AA 4  or hand = AA suited  or hand = A suited 2  or hand = A suited 3 4  or hand = A suited 3 5  or hand = A suited 3 6  or hand = A suited 3 4 5  or hand = A 2 3  or hand = A 2 4  or hand = A 2 5  or hand = A 3 K suited 4 ) fold force

When OpponentsAtTable <= 4 and In SmallBlind and raises = 0
  When calls = 0  and raises = 0  and ( hand = A  or hand = KK  or hand = QQ  or hand = 2 3 4  or hand = 2 3 5  or hand = 2 4 5  or hand = 2 4 6  or hand = K suited Q J T  or hand = K suited Q J 9  or hand = K suited Q T 9  or hand = K suited J T 9  or hand = K Q suited J T  or hand = K Q suited J 9  or hand = K Q suited T 9 ) RaiseMin force
  When raises = 0  and calls >= 1  and ( hand = A  or hand = KK  or hand = QQ  or hand = 2 3 4  or hand = 2 3 5  or hand = 2 4 5  or hand = 2 4 6  or hand = K suited Q J T  or hand = K suited Q J 9  or hand = K suited Q T 9  or hand = K suited J T 9  or hand = K Q suited J T  or hand = K Q suited J 9  or hand = K Q suited T 9 ) call force



Flop

When opponents = 1 and OpponentIsAllIn and BetSize < 20% StackSize call force

When HaveNutLowDraw and HaveNutFlushDraw RaisePot force

When (not LowPossible) and (not FlushPossible) and (not StraightPossible) and HaveNutFlushDraw and HaveStraightDraw RaisePot force

When TripsOnBoard
  When HaveQuads call force
  When (NutFullHouseOrFourOfAkind > 3 or NutFullHouseOrFourOfAkind = 0) fold force

When StackSize <= 8
  When (BotsLastAction = raise or BotsLastAction = bet) RaisePot force
  When not (In BigBlind or In SmallBlind) RaisePot force

When StackSize > 75 and Betsize > 10
  When PairOnBoard and not (HaveFullHouse or HaveNutLowDraw) fold force
  When (not PairOnBoard) and FlushPossible and not (HaveNutFlush or HaveNutLow or HaveNutLowDraw) fold force
  When (not PairOnBoard) and (not FlushPossible) and StraightPossible and not (HaveNutStraight or HaveNutLow or HaveNutLowDraw) fold force
  When (not PairOnBoard) and (not FlushPossible) and (not StraightPossible) and not (HaveSet or HaveNutLow or HaveNutLowDraw or HaveNutFlushDraw or HaveNutStraightDraw or have10outstraightdraw) fold force

When opponents <= 2 and BetSize >= 40% StackSize
  When (not PairOnBoard) and Have2ndNutFlush RaisePot force
  When PairOnBoard and (HaveTrips and Overcards >= 2) RaisePot force

When opponents = 1
  When (not PairOnBoard) and Have2ndNutFlush RaisePot force
  When PairOnBoard and (HaveTrips and Overcards >= 2) RaisePot force
  When PairOnBoard and (HaveTrips and hand = A) and (not board = AA) RaisePot force
  When raises >= 1 and HaveNutLow and not (HaveFlush or HaveStraight or HaveSet or HaveTwoPair or HaveFullHouse or HaveStraightDraw or HaveFlushDraw or HaveTopPair) call force

When LowPossible and (not WheelPossible) and (not PairOnBoard) and (not FlushPossible) and not (board = 236 or board = 246 or board = 256 or board = 346 or board = 356 or board = 347 or board = 357 or board = 367 or board = 457 or board = 467) and opponents = 1
  When HaveLow and not (HaveNutLow or Have2ndNutLow or HaveStraight) and (HaveTwoPair or HaveSet) and bets = 0 and raises = 0 BetPot force
  When HaveLow and not (HaveNutLow or Have2ndNutLow or HaveStraight) and (HaveTwoPair or HaveSet) call force

Turn

When opponents = 1 and OpponentIsAllIn and BetSize < 20% StackSize call force

When (not LowPossible) and (not FlushPossible) and (not StraightPossible) and HaveNutFlushDraw and HaveStraightDraw and HaveNutLowDraw RaisePot force

When (not LowPossible) and (not FlushPossible) and (not StraightPossible) and HaveNutFlushDraw and HaveStraightDraw and BetSize < 50% StackSize call force

When TwoPairOnBoard and (not HaveNutLowDraw) and (NutFullHouseOrFourOfAkind > 3 or NutFullHouseOrFourOfAkind = 0) fold force

When TripsOnBoard and (not HaveNutLowDraw) and (NutFullHouseOrFourOfAkind > 3 or NutFullHouseOrFourOfAkind = 0) fold force

When QuadsOnBoard and not (hand = AA or hand = KK) fold force

When HaveNutLow
  When opponents >= 3 RaisePot force
  When (HaveFlush or HaveStraight or HaveSet or HaveTrips or HaveQuads or HaveFullHouse or HaveTwoPair or HaveTopPair) RaisePot force

When StackSize <= 8
  When (BotsLastAction = raise or BotsLastAction = bet) RaisePot force
  When not (In BigBlind or In SmallBlind) RaisePot force

When StackSize > 75 and BetSize > 15
  When PairOnBoard and not (HaveFullHouse or HaveNutLow) fold force
  When (not PairOnBoard) and FlushPossible and not (HaveNutFlush or HaveNutLow) fold force
  When (not PairOnBoard) and (not FlushPossible) and StraightPossible and not (HaveNutStraight or HaveNutLow) fold force
When (not PairOnBoard) and (not FlushPossible) and (not StraightPossible)
  When HaveTwoPair and HaveNutLowDraw and bets = 0 BetPot force
  When HaveTwoPair and HaveNutLowDraw and StackSize > 25 and bets = 1 call force
  When HaveTwoPair and have2ndnutlowdraw and StackSize > 25 and raises = 0 call force
  When StackSize > 25 and BetSize > 8 and not (HaveSet or HaveNutLow or HaveNutFlushDraw or have12outstraightdraw) fold force

When opponents <= 2 and BetSize >= 40% StackSize
  When PairOnBoard and (HaveTrips and Overcards >= 2) RaisePot force

When opponents = 1
  When (not PairOnBoard) and Have2ndNutFlush RaisePot force
  When PairOnBoard and (HaveTrips and Overcards >= 2) RaisePot force
  When raises >= 1 and HaveNutLow and not (HaveFlush or HaveStraight or HaveSet or HaveTwoPair or HaveFullHouse or HaveStraightDraw or HaveFlushDraw or HaveTopPair) call force

When LowPossible and (not WheelPossible) and (not PairOnBoard) and (not FlushPossible) and not (board = 236 or board = 246 or board = 256 or board = 346 or board = 356 or board = 347 or board = 357 or board = 367 or board = 457 or board = 467) and opponents = 1
  When HaveLow and not (HaveNutLow or have2ndnutlow or HaveStraight) and (HaveTwoPair or HaveSet) and bets = 0 and raises = 0 BetPot force
  When HaveLow and not (HaveNutLow or have2ndnutlow or HaveStraight) and (HaveTwoPair or HaveSet) call force

River

When opponents = 1 and OpponentIsAllIn and BetSize < 20% StackSize call force

When TwoPairOnBoard and not (HaveNutLow or have2ndnutlow) and (NutFullHouseOrFourOfAkind > 3 or NutFullHouseOrFourOfAkind = 0) fold force

When TripsOnBoard and not (HaveNutLow or have2ndnutlow) and (NutFullHouseOrFourOfAkind > 3 or NutFullHouseOrFourOfAkind = 0) fold force

When QuadsOnBoard and not (hand = AA or hand = KK) fold force

When HaveNutLow
  When opponents >= 3 RaisePot force
  When (HaveFlush or HaveStraight or HaveSet or HaveTrips or HaveQuads or HaveFullHouse or HaveTwoPair or HaveTopPair) RaisePot force

When StackSize <= 8
  When (BotsLastAction = raise or BotsLastAction = bet) RaisePot force
  When not (In BigBlind or In SmallBlind) RaisePot force

When LowPossible and (not WheelPossible) and (not PairOnBoard) and (not FlushPossible) and not (board = 236 or board = 246 or board = 256 or board = 346 or board = 356 or board = 347 or board = 357 or board = 367 or board = 457 or board = 467) and opponents = 1
  When HaveLow and HaveTopPair and HaveBestKicker RaisePot force
  When HaveLow and HaveSet RaisePot force
  When HaveLow and HaveTwoPair RaisePot force
  When HaveLow and HaveStraight RaisePot force
  When HaveLow and (not HaveNutLow) and HavePair and BetSize < 50% StackSize call force
When opponents = 1 and (not PairOnBoard)
  When Have2ndNutFlush RaisePot force
  When raises >= 1 and HaveNutLow and not (HaveFlush or HaveStraight or HaveSet or HaveTwoPair or havepair) call force

When opponents = 2 and (not PairOnBoard)
  When (not FlushPossible) and (HaveTwoPair and HaveTopPair) and HaveLow and raises = 0 call force
  When (not FlushPossible) and (HaveSet or HaveStraight) and HaveLow and raises = 0 call force
  When HaveFlush and HaveLow and raises = 0 call force

When StackSize < 20 and potsize > 10 and PairOnTurn and NoBettingOnTurn
  When (not FlushPossibleOnTurn) and HaveNutFlush RaisePot force
  When (not StraightPossibleOnTurn) and (not FlushPossible) and HaveNutStraight RaisePot force
```

PPL

POKER PROGRAMMING LANGUAGE

bonus bots .com